REPORT DOCUMENTATION PAGE

a Report Security Classification: Unclassified		1b Restrictive Markings	
a Security Classification Authority		3 Distribution/Availability of Report	
b Declassification/Downgrading Schedule		Approved for public release; distribution is unlimited.	
Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
a Name of Performing Organization Naval Postgraduate School	6b Office Symbol *(if applicable)*	7a Name of Monitoring Organization Naval Postgraduate School	
c Address *(city, state, and ZIP code)* Monterey CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey CA 93943-5000	
a Name of Funding/Sponsoring Organization	6b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number	
ddress *(city, state, and ZIP code)*		10 Source of Funding Numbers	
		Program Element No / Project No / Task No / Work Unit Accession No	

11 Title *(include security classification)* AN EXPERIMENTAL COMPARISON OF CLOS AND C++ IMPLEMENTATIONS OF AN OBJECT-ORIENTED GRAPHICAL SIMULATION OF WALKING ROBOT KINEMATICS

12 Personal Author(s) Davidson, Sandra Lynne

13a Type of Report	13b Time Covered From To	14 Date of Report *(year, month, day)*	15 Page Count
Master's Thesis		93 / Mar / 25	147

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
ield	Group	Subgroup	Object-Oriented Programming, Walking Robots, Kinematics, Simulation, CLOS, C++

19 Abstract *(continue on reverse if necessary and identify by block number)*

The ability to conduct research in the robotic field in new areas can be accomplished safely and efficiently using computer raphic simulation. Object-oriented languages provide a powerful and flexible capability in defining rigid body manipulators that an be adapted in the use and design of many types of systems. The very nature of object-oriented programming permits odification and improvement of the code with ease.

This thesis examines the major capabilities of object-oriented programming in conjunction with kinematic equations that imulate a six-legged walking robot. A comparison is conducted between programs using CLOS (LISP) and C++ to graphically imulate the Aquarobot - an existing underwater walking robot. It is found that both languages are effective, but CLOS rogramming is easier while C++ code executes more than twice as fast as compiled CLOS.

) Distribution/Availability of Abstract unclassified/unlimited __ same as report __ DTIC users	21 Abstract Security Classification Unclassified	
2a Name of Responsible Individual Robert B. McGhee	22b Telephone *(include Area Code)* (804) 656-2026	22c Office Symbol CS/MZ

) FORM 1473,84 MAR 83 APR edition may be used until exhausted security classification of this page
All other editions are obsolete Unclassified

Approved for public release; distribution is unlimited.

An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Walking Robot Kinematics

by

Sandra Lynne Davidson
Lieutenant, United States Navy
B.S., United States Naval Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1993

ABSTRACT

The ability to conduct research in the robotic field in new areas can be accomplished safely and efficiently using computer graphic simulation. Object-oriented languages provide a powerful and flexible capability in defining rigid body manipulators that can be adapted in the use and design of many types of systems. The very nature of object-oriented programming permits modification and improvement of the code with ease.

This thesis examines the major capabilities of object-oriented programming in conjunction with kinematics equations that simulate a six-legged walking robot. A comparison is conducted between programs using CLOS (LISP) and C++ to graphically simulate the Aquarobot - an existing underwater walking robot. It is found that both languages are effective, but CLOS programming is easier while C++ code executes more than twice as fast as compiled CLOS.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The writer wishes to recognize the diligent guidance, absolute attentiveness, and all encompassing support of my advisor, Professor Robert B. McGhee. Mr. Charles Lombardo's assistance in debugging code and Professor Sehung Kwak's tutelage in CLOS and object-oriented programming concepts were vital to this author's thesis completion. My appreciation also goes to Professor Kanayama, Kenji Suzuki, and Chuck Schue for their work in the research group. Professor Williamson also provided valuable technical advice on C++ issues.

Finally, I am grateful to Eric and Cody for their patience, understanding, and continuous support during my studies.

This thesis was supported in part by the National Science Foundation under Grant BCS-9109989.

I. INTRODUCTION

A. GOALS

The goal of this thesis is to investigate a method for producing a graphic simulation of a walking robot constructed from serial manipulators acting as legs. The main intent is to compare object-oriented code that is based on kinematics using two programming languages, CLOS and C++. This thesis discusses and provides examples of steps necessary for the evolution of a first stage graphic simulator of a walking robot. The walking robot in question is a six-legged underwater vehicle, called "Aquarobot", that is presently under development in Japan for use in subsea construction and inspection tasks.

B. ORGANIZATION

Chapter II of this thesis reviews previous work in the area of walking robots. Chapter III provides a detailed description of Aquarobot, the subject of the simulator developed in this research. Chapter IV provides an overview of kinematics modelling of articulated rigid bodies, and methods used to calculate link parameters for such systems. The last part of this chapter provides the specific kinematic parameters for Aquarobot.

Chapter V is a review of object-oriented programming and includes a discussion of its advantages and disadvantages. Chapter VI contains the history and a description of some common object-oriented languages. Chapter VII provides a description of the Aquarobot simulation programs written in the CLOS and C++ languages. This chapter compares the methods each language requires to define classes and create objects. A comparison of the performance of the C++ and CLOS simulations is provided in Chapter VIII.

The last chapter, Chapter IX, presents some conclusions about the work described. This is followed by recommendations for possible future use of Aquarobot, the characteristics of the two simulations created, and suggestions for further research.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Man's need to comprehend the human body and the phenomena around him motivates him to imitate it as a tool of understanding. This chapter provides a historical review of robotic advancements in living animal imitation. It specifically addresses the evolution of legged robots. The differences between legged and wheeled locomotion are also discussed.

B. HISTORICAL IMITATION OF LIVING CREATURES

Historically, research has attempted to build machines that imitate animals. Through technology, it is hoped to achieve a better understanding of humans and animals and to accomplish these creature's tasks with robots. Some such research is driven by a desire to provide the disabled with alternative compensation methods, such as artificial limbs, and other means of achieving increased mobility (McGhee, 1977). Mobility goals for legged vehicles include moving faster or for extended times, or operating in adverse environments and conditions such as moving under water, and in space flight applications.

Biological systems, often taken for granted, are extremely difficult to emulate or even define. One example, the

imitation of a walking gait of an animal, is not easy due to the difficulty of emulating the nervous system and the natural materials that form the animal. These unknown variables have impeded our success in obtaining the coordination algorithms of even simple animals (McGhee, 1985). A human takes approximately one year to learn how to walk yet, after decades of research, walking machines are still considered to be in the "infant" stage.

Animal limb imitation has been an area of great interest to researchers interested in advanced mobility systems. If an application for a walking vehicle is known, there are many variables that must be considered to determine an animal to imitate. As an example, one variable is compliance (Anon, 1987). Compliance is defined as "the act of conforming, acquiescing, or yielding" (Stein, 1979). As the degree of compliance of a design is improved, the machine becomes more challenging to control and keep the limb steady, yet it will be more robust (e.g., able to withstand impact). If the degree of compliance in a design is reduced, then the ability to accurately position the limb will be enhanced, but it will tend to be rigid and unyielding. On the other hand, compliance permits flexibility which is beneficial when performing simple but complex actions such as attempting to place a bolt on a screw (Anon, 1987).

Limb imitation designs have varied drastically in appearance. For access to tight spaces, snake-like devices

have been constructed. Their applications require that compliance be limited in order to maintain position. In contrast, a limb similar to an elephant's trunk has been used as a device to lift objects of varied shapes. This device did not have an internal support structure. Instead, it copied the multiple layers of muscle in an elephant's trunk which provides motion control. It was extremely compliant in order to accommodate the varied shapes that grasped objects require (Anon, 1987). Human hands have been imitated in numerous designs. Additionally, legs are very popular in robot research.

Legged locomotion requires a successful leg design. Through evolution, animals have perfected their individual legged locomotion characteristics based on their specialized needs. Legged animals are capable of high speeds and intricate motion even when the animal is large and heavy. Animal legs have been put into two categories: "mammal" and "insect" types (Iwasaki, 1987). The "mammal" type has legs which are always vertical like a horse. The "insect" type has bent legs like a beetle. A walking capability able to function in natural terrain requires complicated sensors, a nervous system, and artificial intelligence (e.g., a reasoning ability). Since exact imitation of these intricate animal systems has not, at present, been achieved, legged vehicle designers must choose other means to solve this coordination control problem (McGhee, 1985).

C. HISTORY OF WALKING ROBOTS

The original legged machines evolved from earth moving and construction vehicles. These devices are known as "climbing hoes" (McGhee, 1985). From 1965 to 1968, a four-legged vehicle, called the "Quadruped Transporter", was constructed by General Electric. This vehicle incorporated a human operator in order to provide the sensing and neural control functions discussed earlier. The operator of this vehicle was provided with one leg control lever for each limb. These levers were attached to the arms and legs of the human operator so that he could control the legs by executing the desired motions with his own limbs. The front legs were controlled by the operator's hands and the rear legs were controlled by the operator's feet. Each control lever had three degrees of freedom: two at the hip and one at the knee. This coordination control system required a high level of operator skill, and only a few mastered its intricacies. Moreover, these operators could only walk the vehicle for a short time due to the complicated multi-degree of freedom coordination problem (McGhee, 1985).

The Quadruped Transporter was designed as a research vehicle and opened the field of vehicular legged locomotion. A hydraulic servo system moved the legs. It successfully walked and displayed impressive obstacle climbing ability.

However, the complexity of the operator motion coordination task severely limited the device's capabilities (McGhee, 1985).

In 1977, a different control method was incorporated into another robot called the Ohio State University (OSU) Hexapod Vehicle. This robot used supervisory control (Ferrell, 1967) in which the operator controlled speed and direction, and a computer coordinated the actual leg motion (Pugh, 1982). The OSU Hexapod Vehicle was a six-legged vehicle with insect type legs (McGhee, 1985). The device was constructed to study and develop gait algorithms. Each leg had three degrees of freedom, each consisting of two links connected by a joint. Each joint had an electric motor and a worm gear (Waldron, 1989). The operator controlled the vehicle with a remote joystick in an indoor laboratory setting.

The successor to the OSU Hexapod Vehicle was completed in 1986 at OSU. It was called the "Adaptive Suspension Vehicle" (ASV) (Waldron, 1986). The ASV was designed for sustained outdoor locomotion on uneven and unmapped terrain. This six-legged robot was the first robot to control its legs by an on-board computer and to carry its own power source in the form of an internal combustion engine (Waldron, 1986). The ASV, like the Quadruped Transformer, includes an onboard human operator. However, the ASV does not require manual coordination of limb motion by the operator (Waldron, 1986). In order for the ASV to operate in unstructured terrain, it

incorporates extensive sensor devices including a laser terrain scanner to provide a three dimensional terrain elevation map for a distance of ten meters in front of the vehicle. This information is used for automatic selection of footholds in rough terrain (Waldron, 1986).

To date, legged vehicles have had limited application success. This is due to the complex leg coordination control problem and a limited understanding of necessary gait algorithms. Also, this situation exists because of limited advances in leg design. Future improvements in agility and speed are anticipated with further progress in understanding of the difficult problem of microcomputer coordination of joint motion (McGhee, 1985).

D. ADVANTAGES OF LEGGED ROBOTS

Legged locomotion has existed for hundreds of millions of years while wheeled locomotion, an invention of man, has been around for only several thousand years (Waldron, 1989). It is interesting that evolution has not produced wheeled biological systems, but then there were no smooth, graded roads before the introduction of wheels. Still, given the elegant results of evolution, one might conclude legged locomotion is inherently superior to wheeled locomotion, at least in natural terrain.

Currently, it is possible to go close to most places of interest on the land surface of the earth by traveling on

roads. This has greatly altered our environment. Still, it takes an off-road wheeled or tracked vehicle to reach the areas in between, and they leave ugly ruts in the soil. If the off-road vehicle were to be a legged vehicle, it would leave only discrete footprints. Furthermore, over half the Earth's land surface (largely, unpopulated areas) is entirely inaccessible to wheeled vehicles (Waldron, 1989) but not to legged vehicles. Legged vehicles have the potential to walk underwater and in surf as well.

Legged locomotion has an advantage over wheeled locomotion when soft ground or slippery surfaces are involved. Wheeled vehicles sink into the ground and must roll out of the resulting depression by relying on shearing forces resulting from friction between wheels and the ground. Legs also sink into the ground but can be lifted vertically (Bekker, 1969) - a maneuver that doesn't impede locomotion.

While wheeled vehicles have proven themselves efficient for long-distance transportation, the path must be relatively smooth and firm. The performance of large mammals shows that it is possible for legged locomotion to also be efficient for long-distance transportation. However, actively coordinated leg motions must be defined by algorithms. These algorithms are presently in an early stage of development (Waldron, 1989).

Legged vehicles may eventually be able to compete with wheeled locomotion in all respects except possibly speed.

However, additional technological advances in theory and materials will be needed before such machines can reach their full potential. The advances in computers in the late 1980's enabled researchers to provide for the leg coordination computations on board a walking vehicle, but researchers are moving slowly in their attempts to provide sufficiently powerful computation algorithms (McGhee, 1985). Over adverse terrain, legged vehicles have the potential to provide higher speed, greater mobility, and less environmental damage. Additionally, legged vehicles can provide more comfort for a human rider. The rough ride wheeled provided by locomotion over rough terrain is detrimental to instruments and cargo on board. In contrast, legged vehicles do not vibrate when travelling over rough terrain (Waldron, 1989). Finally, several studies have shown that legged vehicles have the potential to provide improved fuel economy in comparison with wheeled vehicles of comparable size (McGhee, 1986).

E. SUMMARY

This chapter provides a survey of previous work relating to and walking machines. It specifically discusses the history of legged vehicle technology and provides walking machine examples. Legged and wheeled locomotion are compared and their specific advantages are discussed. The next chapter

discusses a walking robot, Aquarobot, that is currently under development in Japan, and which provides the focus of this thesis.

III. AQUAROBOT

A. INTRODUCTION

One of Japan's most important resources is its land. Unfortunately, Japanese tidal waves (tsunamis), constantly threaten the Japanese coast and erode productive ground. Granite rock mound foundations are currently being laid for a tsunami seawall to be installed in Kamaishi Bay in the northern part of Honshu. This seawall is designed to dissipate the energy of a tsunami prior to its arrival at shore. The Port and Harbour Research Institute (PHRI) of the Ministry of Transportation in Yokosuka, Japan, wishes to develop a general method to accomplish deep water structural inspection of seawalls, including the Kamaishi project. This method should also provide supervision of construction and quality control, and should not involve the use of human divers (Akizono, 1989).

Unfortunately, there is not an "optimal" device to accomplish the task that PHRI requires. PHRI is currently using human divers to measure wall and foundation variations. This is a difficult process due to the pressurization requirements of the human body and the short time that divers can be at the bottom (about one hour per day at a depth of sixty meters). Additionally, the deep sea diver occupation is

physically taxing and it is difficult to recruit new personnel. At this time, most of Japan's deep sea divers are in their late 30's or older (Takahashi, 1993). Human divers are very capable when at the depth of the wall, but are slow and expensive.

Using a robot is one obvious alternative. There are two basic options in the design of such a robot. First, a floating Remotely Operated Vehicle (ROV) could be used. However, floating vehicles have difficulty maintaining a stationary position while keeping a specified heading in water. A floating vehicle has a poor ability to accurately measure objects since the vehicle is not stable. This would make a floating robot a poor choice for the PHRI measurement needs. However, a floating vehicle is an excellent choice for camera inspection because it can move a TV camera to all viewing aspects. Unfortunately, if the sea floor is muddy, a floating robot may make the water murky due to turbulence induced by its thrusters used for maneuvering (Robison, 1992).

Another robot type available is the walking robot. It provides stability in a stationary position. It can provide the measurements PHRI desires. However, there will be limitations on the camera angles dependent upon the degrees of freedom of the camera arm and the arm placement. A walking robot does not muddy the water because it does not stir up a soft sea floor. Of the two general types of walking robots, "mammal" and "insect", the insect type provides better

movement on uneven terrain (Waldron, 1989). Aquarobot is an insect type walking robot.

B. AQUAROBOT HISTORY

PHRI has designed three robots in an attempt to produce the first practical underwater walking robot. These robots have been labeled "Aquarobot" by their creator, PHRI (Akizono, 1989). They are all six-legged articulated robots.

The first, an experimental model, was designed in 1985. It was not watertight and was designed to run ground tests for basic research and as a software debugger.

The second Aquarobot, the prototype model, was designed for underwater sea floor applications. The third Aquarobot was designed as a lightweight design of the prototype model. It is the second prototype model which has been modeled in this thesis (Akizono, 1989).

C. DESCRIPTION

The prototype Aquarobot is a walking ROV designed to follow a path determined from navigation beacons using a gait algorithm computed by a control station on a barge on the surface, and passed to the robot via a tether. It is a six-legged articulated "insect type" robot equipped with one arm used to move and aim a video camera (Akizono, 1989).

The aquarobot consists of a hexagonal body and six legs. The body is constructed of anti-corrosive aluminum. Each leg

has three rotary joints that provide three degrees of freedom. Additionally, each leg has a disc-shaped foot pad that is attached to the leg with a freely rotating ball joint. The foot pads are not position controlled, but are oriented by a combination of gravity, the terrain surface, and hydrodynamic effects acting on the ball joint connection. Figure 2.1 depicts the Aquarobot and its leg structure.

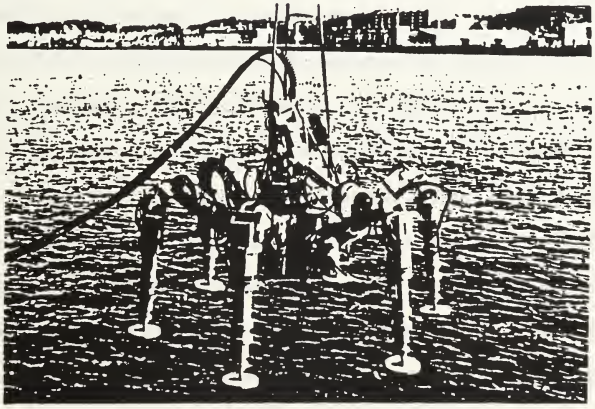


Figure 3.1

Photograph of Aquarobot

Each leg joint of Aquarobot is controlled by the computer via a DC motor that drives a reduction gear. The reduction gear consists of a harmonic gear and a pair of beveled gears. This drive method is known as a semi-direct drive mechanism (Akizono, 1989). These motors and gears are located within the legs.

Each of the eighteen motors are driven by DC power. There is one motor driver per motor, each located on the barge controlling Aquarobot. The motor driver sends the motor a voltage computed from pulse information it receives. Specifically, the motor driver contains a pulse counter which counts up for pulses received from the computer and counts down when pulses are received from encoded motor output feedback. The motor driver provides the necessary voltage to the motor to drive the counter toward zero. Thus, the motor/driver system uses a simple position feedback method. (Akizono, 1989)

There are two inclinometers and one gyrocompass (Anon, 1992) on the body of Aquarobot. Each foot has a pressure sensitive touch sensor. These sensors provide foot contact and body orientation information to the computer (Akizono, 1989). To measure the absolute elevation of selected points on a rock mound foundation, one leg of Aquarobot is also furnished with an accurate depth cell located just above the foot (Takahashi, 1993).

The computer system, located on the barge, provides walking algorithms and operating programs. It is a 16-bit controller. The interface is provided by two integrated circuit boards: an input/output board and an A/D converter board. The input/output board sends pulses to the motor driver and receives touch sensor status and joint rotation pulses from the legs. The A/D converter receives the

gyrocompass, depth cell, and inclination sensor feedback. Individual leg motions thus are performed using hardware controls, while top level motion control and path planning is controlled by software. (Akizono, 1989)

The information bus has changed throughout Aquarobot's evolution. The tether for the experimental model consisted of copper wire. The two later models have optical fiber links with optical/electric converters in the body and control unit. However, all models contain eighteen copper wires to carry current to individual motors, resulting in a rather large cable cross section (four centimeters). (Iwasaki, 1987)

The computer software is currently written in BASIC. The operating program receives the walking commands from the gait algorithm and simultaneously translates them to the motor drivers in pulse form.

The prototype model's video camera arm has three rotary joints. Cameras may also have independent pan and tilt control. The arm is also equipped with an ultrasonic ranging device. Using this device, scales can be projected on the camera screen so that measurements of an object can be interpreted in conjunction with its range from Aquarobot to determine actual dimensions.

The prototype model also has a relative navigation capability which uses a transponder system. This system

measures its position in cartesian coordinates, based upon triangulation of signals received from beacons placed in the vicinity of Aquarobot at known locations. (Akizono, 1989)

D. CURRENT USE IN JAPAN

The prototype Aquarobot has successfully walked underwater. It's current maximum walking speed on uneven sea bed is approximately one meter per minute. While this speed is judged to be acceptable, Aquarobot has not been put to practical use because human divers are still able to perform its function at a lower cost. (Takahashi, 1993)

E. POSSIBLE AQUAROBOT IMPROVEMENTS

Aquarobot could be improved in many ways. The physical characteristics of the tether and the resultant effects of currents on it is an area where substantial improvements are possible. The tether could be decreased from its currently large circumference and bulky appearance. This could be done by improving the motor controllers and placing them in the vehicle. In this way, the eighteen wires in the cable carrying motor currents could be replaced by a single two conductor power cable. Additionally, the computer software could be optimized to provide faster and more flexible code. New technology in integrated circuits should be incorporated to generally decrease component size and power requirements.

F. SUMMARY

Aquarobot represents a major advancement in the field of walking robots. Aquarobot's design was influenced by the mission it was to accomplish. This is not often the case in robot design. Usually, a robot is designed from a research standpoint and then may be used in a "real life" application. When an application is driving the technology, robotics advancement looks at the problem from a new perspective and new and varied designs can be anticipated. The algorithms required to calculate the leg and body positions of Aquarobot are described in the next chapter of this thesis.

IV. KINEMATICS MODEL

A. INTRODUCTION

Robots typically consist of one or more "limbs" which are technically defined as mechanical manipulators. These manipulators provide the robot with the capability to grasp, walk, or perform some other task. To control the robot appendages with commands to move to a desired location, knowledge from the field of physics and engineering that describes motion of rigid bodies is needed. This field is known as kinematics. Kinematics is "... the science of motion which treats motion without regard to the forces which cause it" (Craig, 1989, p.6). Kinematics allow all geometric properties of the motion to be defined.

Forward kinematics computes the Cartesian space position and orientation of the manipulator links from a set of parameters which describe the manipulator using angles and lengths. The orientation is often described as azimuth, roll, and elevation. Inverse kinematics solves for the manipulator parameters when the Cartesian space and orientation are known.

B. LINKAGE AND COMPONENT DESCRIPTION

Manipulators consist of nearly rigid links which are connected at joints. There are two simple types of joints: sliding (prismatic) and rotary. The joints are designated by

number beginning from the base, usually labeled joint 0 (Craig, 1989). The base is also sometimes considered to be the most inboard link. The free end of the links is the most outboard link and is often called the end-effector. It is at the end-effector that the robot's work is performed. Often the end-effector is a grasping device or a foot pad.

Kinematics considers each link to be a purely rigid body (Craig, 1989). In reality, description of a manipulator's links requires many variables to be considered during the design process. Some variables include the material used for construction, the link strength, stiffness, length, and the manipulator weight.

Kinematic algorithms are designed to define the position and orientation of all manipulators regardless of their geometric complexity. This is accomplished by carefully defining joint coordinate axes called frames and arranging their alignments using standard parameters that describe the adjacent link relationships (Craig, 1989).

C. KINEMATICS PARAMETER DEFINITIONS

A frame is attached to each joint with the Z-axis coincident with the joint motion axis. The X-axis of the frame is directed from a link's inboard joint towards its outboard joint to intersect that joint's axis, and is mutually perpendicular to both Z-axes.

Four parameters are needed in the kinematic algorithms. The first, link length, is the distance along the X-axis between the joints of a given link. The second is link twist. This is the angle necessary to rotate the inboard Z-axis to be parallel to the outboard Z-axis.

The third parameter is link offset. It is the distance measured at the inboard link axis from the preceding link X-axis to the current X-axis. The final parameter is the rotation at this joint from the previous link X-axis to the current link X-axis. This is known as the joint angle.

D. CRAIG VERSUS DANEVIT-HARTENBERG METHOD COMPARISON

Forward kinematics determines the cumulative effect of joint motions on the entire link chain. This cumulative effect can be accomplished by a number of methods. Two common methods, Craig and Danevit-Hartenberg, are related in their approach but differ in their setup (Spong, 1989).

To begin with, the manipulator must be inspected. The frames must be placed with the proper orientation. The four parameters discussed above must then be determined. These parameters are identical for both methods; however, the numbering of the joint frames varies.

The Craig method numbers the links beginning with zero at the most inboard link. The base joint is numbered joint 0. This produces a numbering system where the link and the link's inboard joint have the same index number (Craig, 1989). An

example of this notation is pictured in Figure 4.1. The base (joint 0) inboard link length and inboard link twist are both defined as zero.

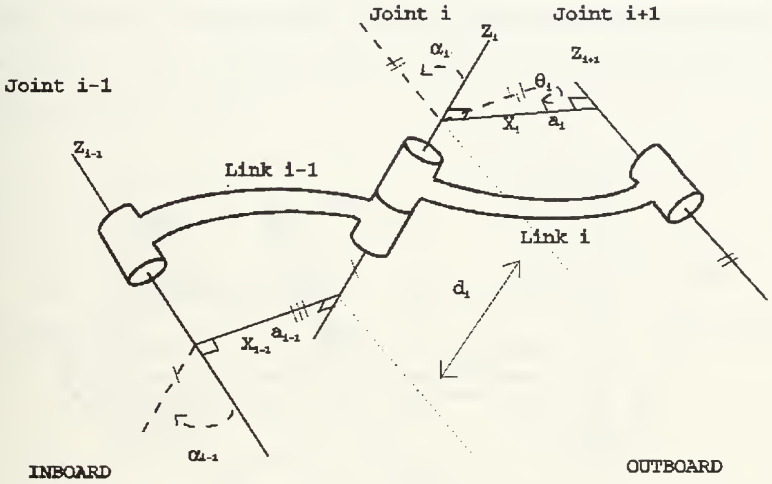


Figure 4.1

Craig Method Frame and Parameter Assignment

The Danevit-Hartenberg (DH) notation differs from the other method. In this method, the first link, attached to the base joint, is labeled link 1. The base joint is labeled joint 0. This produces a numbering system along the link chain in which the link and the link's outboard joint have the same index number (Spong, 1989). An example of this method is pictured in Figure 4.2.

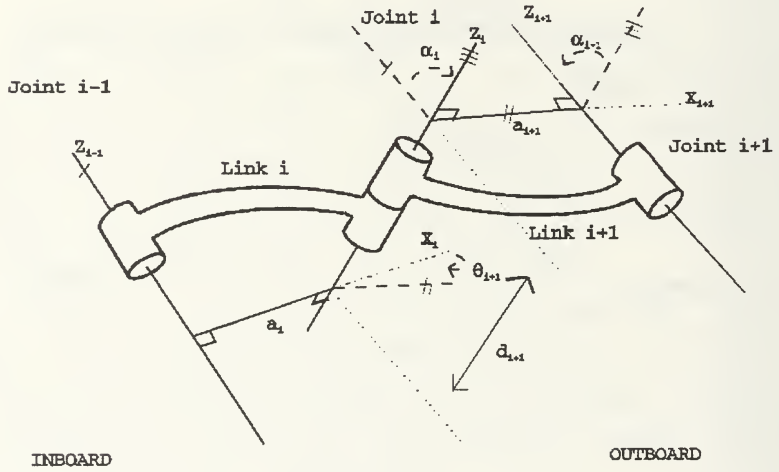


Figure 4.2

Danevit-Hartenberg Frame and Parameter Assignment

These methods use related conventions for manipulating these parameters; however, the algorithms are different. The cumulative effect of the links are defined within a matrix known as the transformation matrix (Craig, 1989). The transformation matrix differs for the two methods addressed.

The Craig method uses a transformation matrix (known as the T matrix) to define the outboard joint location on a link relative to the inboard joint. The T matrix is defined as (Craig, 1989, p.84):

$$
{}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i\, c\alpha_{i-1} & c\theta_i\, c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}\, d_i \\ s\theta_i\, s\alpha_{i-1} & c\theta_i\, s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}\, d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.1)
$$

where $c = \cos$ (4.2)

$s = \sin$ (4.3)

The subscript of the T matrix label describes which joint is being defined. The superscript of the T matrix describes the link to which the matrix is referenced.

The Danevit-Hartenberg method uses a transformation matrix (known as the A matrix) to define the location of the inboard joint on a link relative to the outboard joint. That is, the coordinate origin for a link is located at its outboard joint for the DH method, while it is at the inboard joint in the Craig method. The A matrix is defined as (Spong, 1989, p.66):

$$
{}^{i}_{i-1}A = \begin{bmatrix} c\theta_i & -s\theta_i\, c\alpha_i & s\theta_i\, s\alpha_i & a_i\, c\theta_i \\ s\theta_i & c\theta_i\, c\alpha_i & -c\theta_i\, s\alpha_i & a_i\, s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.4)
$$

The subscript and superscript of the A matrix are defined the same as the T matrix above. However, by convention, the index is transposed.

These transformation matrices provide information on the rotation and translation needed to superimpose the frame being transformed to the relative frame. The rotation information is the top left 3 x 3 sub matrix in the transformation matrix. The translation information is in the right column in the first three rows.

The base joint is aligned with the coordinates that the designer would like to use to reference the link positions. Usually, for fixed base manipulators, the base joint axis is aligned with the Earth's coordinates. To transform the joint in question, the transformation matrices need to be multiplied together (Craig, 1989). For example:

$$ {}_{4}^{0}T = {}_{1}^{0}T * {}_{2}^{1}T * {}_{3}^{2}T * {}_{4}^{3}T \qquad (4.5) $$

$$ {}_{0}^{4}A = {}_{0}^{1}A * {}_{1}^{2}A * {}_{2}^{3}A * {}_{3}^{4}A \qquad (4.6) $$

E. AQUAROBOT KINEMATICS

Aquarobot's six legs are identical manipulators except for their angle off of the body's forward axis. In order to simplify the leg parameters of the first link, an imaginary link was constructed from the body's center to the point where the leg joins the body. This makes the body's center the base joint. The Craig method will be used in this thesis to solve the kinematic equations for Aquarobot's legs.

1. Aquarobot Leg Parameters

Common symbols exist for the parameters. They are: link length (a_i), link twist (α_i), link offset (d_i), and joint angle (θ_i). Figure 4.3 shows one Aquarobot leg with the imaginary leg link included.

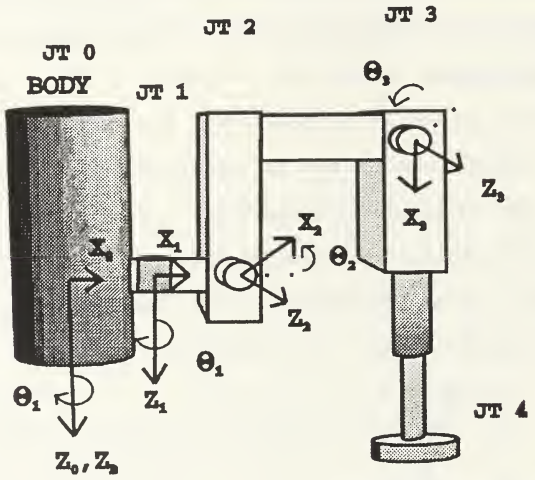


Figure 4.3

Aquarobot Frame Descriptions Of One Leg and The Body

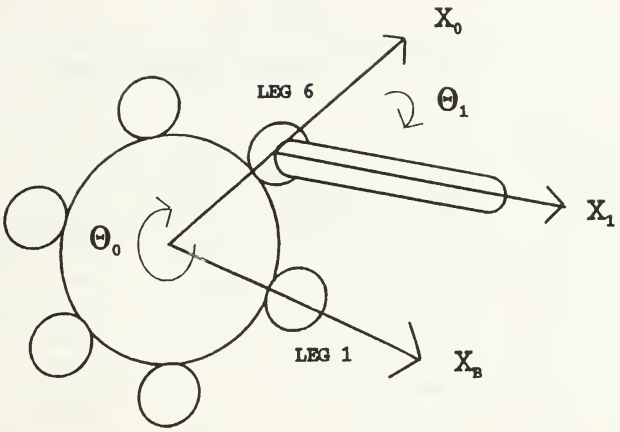


Figure 4.4

Top View of Aquarobot Showing First Two Angles and Axes for Leg Six

The parameters for Aquarobot's legs are shown in Table 4.1 below. Note that the joint angle of the base (i = 0) is the only fixed parameter that varies among the legs. The joint angle range for the other joints common to all legs are given. These limits are the physical joint ranges. Joint four does not have a frame designated because it is a passive ball joint.

TABLE 4.1

AQUAROBOT KINEMATICS PARAMETERS

joint index i	inboard link twist angle α_{i-1}	inboard link length a_{i-1}	outboard link offset d_i	outboard link joint angle θ_i	physical limits
0	0.0	0.0	0.0	θ_0	$\theta_0 = 0, 60, 120, 180, 240, 300$
1	0.0	37.5	0.0	θ_1	$-60 <= \theta_1 <= 60$
2	-90.0	20.0	0.0	θ_2	$-106.6 <= \theta_2 <= 73.4$
3	0.0	50.0	0.0	θ_3	$-156.4 <= \theta_3 <= 23.6$
4	0.0	100.0	0.0	θ_4	$-45 <= \theta_4 <= 45$

2. Transformation Matrices

The transformation matrices of the Aquarobot legs were constructed using Table 4.1 above. The Craig method transformation matrix template was used. The computed link transformations are thus:

$$ {}^{B}_{0}T = \begin{bmatrix} c\theta_0 & -s\theta_0 & 0 & 0 \\ s\theta_0 & c\theta_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.7) $$

$$ {}^{0}_{1}T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 37.5 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.8) $$

$$
{}^{1}_{2}T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 20 \\ 0 & 0 & 1 & 0 \\ -s\theta_2 & -c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.9)
$$

$$
{}^{2}_{3}T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & 50 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.10)
$$

$$
{}^{3}_{4}T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & 100 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.11)
$$

Multiplying the T matrices together provides the joint coordinates in reference to the body's center. These Cartesian coordinates are found in the third column of the product of the T matrix multiplication.

The T matrix product of each joint is called the Homogeneous Transformation matrix (i.e., H matrix). To determine the next outboard joint's orientation based upon the reference frame, the H matrix of the current (relatively inboard) joint is multiplied by the outboard joint's T matrix. The joint's H matrix provides the orientation from the reference frame outboard to that joint.

The H matrix for the body provides orientation of the body frame (and, in turn, its outboard joints) to the fixed reference frame which is usually a designated point on Earth. The initial orientation information required is azimuth, elevation, roll, and translation from the reference's origin. Elevation is defined as rotation of the body X-axis above or below the horizontal plane. Azimuth is the rotation of this axis away from north about a downward directed axis. Roll is rotation about the body X-axis after azimuth and elevation rotations have been accomplished. The H matrix is defined as (Craig, 1989, p. 46):

$$H = \begin{bmatrix} c(a)c(e) & c(a)s(e)s(r)-s(a)c(r) & c(a)s(e)c(r)+s(a)s(r) & x \\ s(a)c(e) & s(a)s(e)s(r)+c(a)c(r) & s(a)s(e)c(r)-c(a)s(r) & y \\ -s(e) & c(e)s(r) & c(e)c(r) & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.12)$$

where a = azimuth e = elevation r = roll

When the body moves, its H matrix relates its body coordinate system to the world coordinate system. The cumulative effect of the body's motion is transferred to the individual links via the H-matrix and continues to be transferred outboard in this manner.

F. INVERSE KINEMATICS

Inverse Kinematics provides the parameter values needed to move the joints to a desired position. The transformation matrix products above are equated to the generic transformation matrix to make a set of nonlinear equations (Craig, 1989, p. 123).

$$T = \begin{bmatrix} \begin{bmatrix} \mathbf{R} \end{bmatrix} & \begin{bmatrix} \mathbf{T} \end{bmatrix} \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.13)$$

These equations are solved simultaneously for the unknown parameters (joint rotation in the case of Aquarobot). The inverse kinematics of Aquarobot are solved in another thesis (Schue, 1993). There are occasions when two solutions for a parameter are possible (Craig, 1989).

G. SUMMARY

Aquarobot's design uses rotating joints. Rotating joints have an advantage over sliding joints in that they generally provide increased dexterity. Additionally, such joints can usually be made smaller than sliding links (Spong, 1989). They are also easier to waterproof for an underwater walking robot.

Kinematics analysis permits Aquarobot's foot positions to be easily determined using successive transformations. Kinematics equations can be manipulated quickly using computers. Object oriented programming simplifies the numerous transformations necessary for an intricate multi-link system. Object oriented programming is discussed in the next chapter.

V. OBJECT ORIENTED PROGRAMMING

A. INTRODUCTION

Object Oriented Programming emphasizes the subjects which operations act upon in contrast to the traditional programming method of emphasizing the algorithms and the order necessary to execute them (Booch 1991). The Object Oriented (OO) designer constructs his modules of code based on items (known as objects). These objects need not in every case accomplish anything significant, but they do at a minimum provide encapsulated data. Other designers construct their modules based upon the data and algorithms that are associated with such blocks.

OO code permits the designer to produce elementary components and then link these objects together to produce a complex system. This parallels the thought process that humans use to think of objects around them.

OO code provides two structures, object and class. Classes are the blueprints of a component and exist in a "kind of" hierarchy. Objects are the actual produced copy of the object (instances of classes) and may exist in a "part of" hierarchy in relation to other objects.

B. CLASS DEFINITION AND CLASS HIERARCHIES

Classes are the building blocks or key designs of a system. They are synonymous with a factory's product blueprints. Classes provide the ability to make many modules (objects) that are designed identically. Each object, when made, provides the "essence" of the class (Fink, 1992). Classes are static, and the information, known as fields, of the class are fixed. The class definition provides a template for the production of objects.

A class can inherit from one or more superclasses. The inheriting class is known as a subclass. A class can also have subclasses which consists of it and additional information. Each senior, top level module, represents one of the most general designs in a system.

Class structures may include object fields, also known as slots, from multiple superclasses with subclasses created using some priority scheme or other means to resolve conflicts (Booch, 1991). These class frameworks are transferred to the objects that are produced.

Multiple class structures form a design. Seldom does one class concisely define a system. The class hierarchy permits all nuances desired, regardless of significance, to be defined as the designer chooses at that level.

The classes designed to serve as templates are defined as concrete classes. They are expected to have objects instantiated from them (de Paula and Nelson, 1991).

Not all classes are designed as templates for actual object instantiation. These are called abstract classes. They are higher level classes which hold knowledge that all of their subclasses have in common. Abstract classes reduce duplication of common knowledge (Wu, 1991). They are written so that multiple subclasses can inherit from them.

A class capability provides two of the three features necessary for OO programming. First, it provides for a design to be defined in a generic format. Second, it allows the designs to be modularized at the most general level, yet still provides for a relationship framework where additional design features can be added in subclasses (Booch, 1991).

C. OBJECT DEFINITION AND OBJECT HIERARCHIES

Objects are the useable products of OOP and provide the third capability needed for utilization of this technology. They are concrete software entities that can be manipulated. An object has all of the properties of its class. All objects produced from the same class contain identical fields and functions, yet it is important to understand that each object has its own identity and its own name when produced. It is by this name that the object is addressed within the code. The objects are facsimiles of the class and its behavior and fields. However, they may be elaborated individually (Eckel, 1989).

An object is constructed by creating an instance of the class desired. All superclasses and their defined functions, also known as methods, are available to the object. It is not necessary to have an object for every class. A class may have zero, one, or multiple instances of itself (de Paula and Nelson, 1991).

An instance of an object may be produced by two different schemes. First, an object may be instantiated within the main program. This object may subsequently be addressed by its created name. Such an object is a top level object within the software. It may also be considered a subobject if it is used as part of a larger composite object. The second type of object, a dependent object, is instantiated directly and automatically as a part of another object. The dependent object is considered a component of the object it is instantiated within and has no independent name. A dependent object is instantiated during construction of the main object and is destroyed when the main object is destroyed. For example, a sports car, when produced, can be thought of as an object and its components, such as the doors, can be considered dependent objects since they are made during the sports car's construction and are legally part of the vehicle.

Objects, when instantiated, acquire all of the fields of the class, but the values of these characteristics may be initialized individually during the construction of the individual objects. For example, this permits numerous

objects of the same class to be instantiated yet have different measurements or characteristics.

An object may change its slot values during its existence. This provides an object with a history. An object may be created and destroyed. The object's functions can not be violated. Objects perform functions by sending requests. These functions are designed within the class structure and are applicable to the objects instantiated from the specific class or its class hierarchy. This capability to perform functions enables an object to be much more than a data structure.

Functions, history, and "lifetime" characteristics provide objects with state, behavior, and identity (Booch, 1991). This parallels their real-world counterparts.

D. INHERITANCE

Inheritance is defined as a "... mechanism for resource sharing in hierarchies" (Wegner, 1987, p. 169). It is a unique contribution of OO languages. Inheritance provides an easy way to create objects that are very similar, although individual instances may have some differences (Stefik, 1986). The subclass is a specialization that augments or alters the structure and behavior of the inherited class. It inherits all functions and methods defined for its superclasses. This includes all attributes that the superclass inherited form its superclass (Wegner, 1987). A subclass may have fields or

methods which modify, elaborate, or add to those inherited by its superclass (Booch 1991).

When a subclass has one superclass, this is called single inheritance. Two or more superclasses defines multiple inheritance. Inheritance is a class relationship rather than an object relationship.

E. CLASS AND OBJECT DIAGRAMS

Class and object diagrams provide a logical view of a system. The difference between object and class diagrams is an important concept in OO design.

Class diagrams are built on interclass relationships involving inheritance. Superclasses and subclasses describe these diagrams. Class utilities provide a special relationship within the class diagram. A class utility is an abstract class type which provides functions that do not belong to one particular class but are accessible to all. Figure 5.1 displays an example of a representative class diagram. Note that there is only one of each class in the class diagram.

Figure 5.1 displays an automobile class diagram. The top level module, the automobile class, is the superclass. In this class structure, subclasses are necessary. Two subclass levels are necessary in order to reach a concrete class. The Porsche can be produced but a sports car is an abstract class that can not be instantiated.

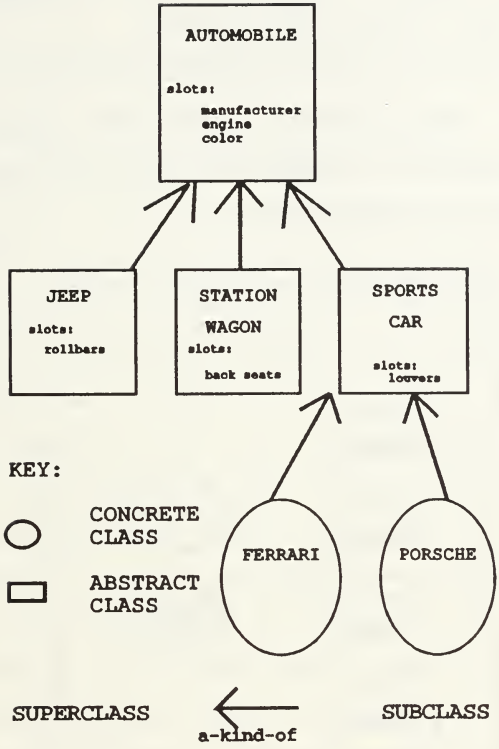


Figure 5.1

Example Class Diagram

Object diagrams "... show the existence of objects and their relationship in the logical design of the system ..." (Booch, 1991, p. 169). An object diagram shows the

relationship between objects and presents each object as "part of" the total system. Figure 5.2 is an example of an automotive system via its object diagram.

In Figure 5.2, a Porsche is instantiated as the object desired. In order to create a Porsche, however, many subsystems (dependent objects) are needed. A few example dependent objects are displayed.

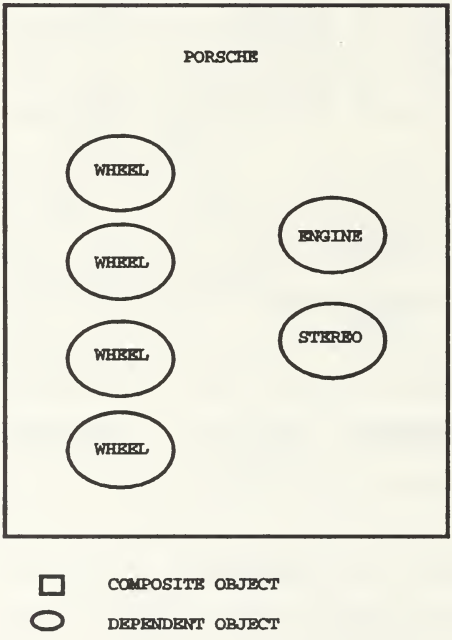


Figure 5.2

Example Object Diagram

A composite object is defined as an object linked to other objects by part-of relationships. Parts of the composite objects may be subobjects or dependent objects.

F. CONCLUSIONS ABOUT OBJECT ORIENTED DESIGN

OO Programming allows designers to begin with a simple or general system. This principle of beginning with a large, less specific class or object is similar to human perception. First, humans determine what the overall item in questions is. For example, a person may look at a car. He or she is likely to note the model of the car at that time. Then, smaller "part of" subsystems of the car or specific slot values may be inspected. For example, the year the car was produced or the air conditioning system may be looked into. OO code permits the programmer to define subclasses or components when they are needed or as the system is elaborated upon. This allows the programming to be accomplished in small increments. OO designs allow attention to be focused on the appearance and external capabilities of objects instead of on software implementation details. This prevents too much information from "cluttering" the user's view (Snyder, 1986). Another advantage to OO Programming is the capability to improve or alter class slots as the system changes or as corrections are needed. This resilience and the capability to reuse small subsystems in multiple objects makes OO code economical (Booch 1991).

OO Programming encourages reuse of entire software class hierarchies. The modular design of OO code permits new users to incorporate existing code without having to retest functions or redesign code. This extendibility of code life reduces a designers programming time. The modular design of class code permits the user to use the functions of the class without requiring an intimate detailed knowledge of the function's inner workings.

A disadvantage of OO Programming is that classes may be designed without placing functions in the most general superclass. This causes identical functions to be defined in numerous subclasses and increases complexity. Repetition should be minimized by placing common functions of two or more classes in a superclass. Of course this can be an iterative process with common properties or methods being factored out and moved upward in a class hierarchy as they are noted.

G. SUMMARY

Human capacity is limited in its capability to grasp complex systems. OO code enables a person to look at a complex system as a collection of various subsystems. It also provides the capability to only look at areas of interest within an object/class.

Object Oriented programming provides software that is "malleable". This is directly due to class structure and inheritance, a unique characteristic of this code (Booch, 1991).

The difference between the class and object structures is a subtle but important one. An object is an instance of a class and the object may be created and destroyed within a program. A class is designed but it is static when a program is executed (Booch, 1991).

The long life span, maintainability, and flexibility in application of Object Oriented code makes it the premiere choice when a design with multiple subsystems is desired. The greatest hindrance of Object-Oriented Programming's potential to be the popular choice in industrial design is its current lack of widely excepted standards. However, there seems to be considerable consensus on OOP's primary concepts. The next chapter describes OO languages suitable for development of an Aquarobot kinematic simulation.

VI. OBJECT ORIENTED PROGRAMMING LANGUAGES

A. INTRODUCTION

Not all computer languages are able to support OOP. Four predominant languages with OO capability are CLOS, C++, Object Pascal, and Smalltalk. Aquarobot is designed using CLOS and C++. Aquarobot's class and object hierarchies are described and then created with CLOS and C++ code.

B. DESCRIPTION OF CLOS

1. History

LISP evolved in the late 1950's and was named for its performance method: List Processing (Winston, 1989). The fundamental element in LISP is a wordlike object known as an atom. A group of atoms (similar to a sentence of words) is known as a list (Winston, 1989). It is these lists which LISP is designed to manipulate. LISP allows for lists to be added to or deleted from indefinitely. Specific atoms may be extracted or manipulated using LISP created or library functions.

Common LISP was officially designed in 1984 to accumulate the existing LISP variations into one standard version. This standardization was advantageous for academic and industrial use (Steele, 1990). Common LISP was then

extended to provide OO capability and this extension is known as CLOS (Common LISP Object System) (Steele,1990).

CLOS (pronounced see-loss) permits each class to have local and shared slots. These slots can be directly accessed and modified by the programmer (Winston, 1989). CLOS provides for multiple inheritance within the class hierarchy. Conflicts in multiple slot inheritance is avoided due to conflict precedences which define the first superclass listed as superior (Fink, 1992).

2. Benefits in the Kinematics Solution

CLOS (and therefore LISP) has many advantages in the robot kinematics solution. CLOS operates in an interpretive environment that facilitates interactive programming, providing information such as variable status, with rapid response (Winston, 1989). This capability for immediate answers to a drafter's questions provides ease in debugging as well as the drafting of programs (Winston, 1989). Lists are addressed and manipulated using programmer defined symbolic names which generally tend to decrease the code length and improve readability (Keene, 1989).

The symbol manipulation and interactive capability of CLOS simplify the kinematics solution. Each limb's joints can be placed in one list or each parameter can be designed as an atom in a joint list. CLOS code provides compact code for extensive systems as well as functions that are easy to read.

More generally, academic institutions and industries can create complex systems with big programs that run faster and are less expensive due to the compact code (Winston, 1989).

CLOS also incorporates the natural language orientation of LISP. Class and object structures and their slots and values are easily understood (Keene, 1989).

C. DESCRIPTION OF C++

1. History

C++ was designed in 1986 at AT&T Bell Laboratories by Bjarne Stroustrup, and is a superset of the C language (Booch, 1991). C++ incorporates the programming abilities of C and adds OO properties as well as type checking and operator overload functions. In 1989, C++ Version 2.0 provided multiple inheritance (Stroustrup, 1991).

2. Benefits in the Kinematics Solution

C++ is similar in format to many presently popular languages such as Ada and C. The familiar format is an advantage of C++ when an OOP language is necessary. Conversely, LISP has a unique format that is not currently a popular choice in academic institutions or industry. This uniqueness is a disadvantage to CLOS unless the drafter understands LISP.

C++ permits users to apply functions without necessitating intimate detailed knowledge of the class inner workings (Booch, 1991). C++ uses a header file to provide a

top level view of class structure and the functions which apply (Booch, 1991). Functions must be defined for a specific class within its hierarchy because of C++'s strong typing. Subclasses may alter functions that their superclass defines. Common operators such as addition (+) and equality (=) are generically defined for common classes (e.g., integer, double, array, etc.), but they must be redefined in new classes where their use is desired (e.g., a matrix class or link chain) (Ammeraal, 1991).

D. SMALLTALK AND OBJECT PASCAL DESCRIPTIONS

Smalltalk and Object Pascal are also OO languages. Like C++ and CLOS, Object Pascal provides an enhancement of the Pascal language. Object Pascal was specifically designed to add an OO capability to Pascal. However, Object Pascal is more restrictive than C++ in code development (Booch, 1991). All class slots are public so slots may be changed while performing another class function (Booch, 1991).

Smalltalk was designed as a pure OO language and provides many predefined classes. Unlike Object Pascal, all slots in Smalltalk are private. Object Pascal is unique in that it provides an overall system template and all created classes are considered subclasses of a predefined superclass called "Object" (Booch, 1991). Smalltalk is not a strongly typed system, therefore a compiler cannot optimize the code. Smalltalk is limited to single inheritance (Booch, 1991).

E. SUMMARY

OO capability is presently a popular (and seemingly necessary) addition to current programming languages. This development coincides with the increased use of OO in academic and industrial system design.

CLOS provides an easier format for user's to read than C++. However, C++ provides non-list manipulations which are often more efficient. Both languages require knowledge of the original language they embellished or a similarly formatted language. CLOS requires less code space than C++, but C++ usually executes more efficiently, and may require less memory.

CLOS provides dynamic memory allocation and uses "garbage collection" to accumulate unused memory space. Activity is suspended during garbage collection which may hinder real time calculations in some garbage collection methods. In contrast, C++ uses a memory heap which requires that memory be removed and then explicitly returned to the heap when the memory space is no longer needed. The next chapter provides a description of CLOS and C++ and examples of their formats in the context of the Aquarobot code developed in this thesis.

VII. AQUAROBOT CODE DESCRIPTION

A. INTRODUCTION

In order to produce an Aquarobot simulation, each of the robot's major parts needed to be simulated. OOP was chosen as the best method to achieve this goal. One version of Aquarobot was written in CLOS by Prof. Robert McGhee at the Naval Postgraduate School. The other version was written in C++ by this author. In this chapter of this thesis, the object and class diagrams for these two inplementations are presented along with examples of the method each language uses to produce an individual class and instantiate an object. The C++ graphics code is discussed and examples of the display are included. The complete CLOS and C++ Aquarobot programs are found in Appendix A and B respectively.

B. AQUAROBOT CLASS AND OBJECT HIERARCHIES

The class hierarchies designed to produce an Aquarobot in CLOS and C++ are shown in Figures 7.1 and 7.2 respectively. These two figures are not identical, but there are major portions that are similar.

The RigidBody class is a superclass of the system. Its subclasses are the major pieces with which Aquarobot and its components are created. The AquaLeg class uses the Link

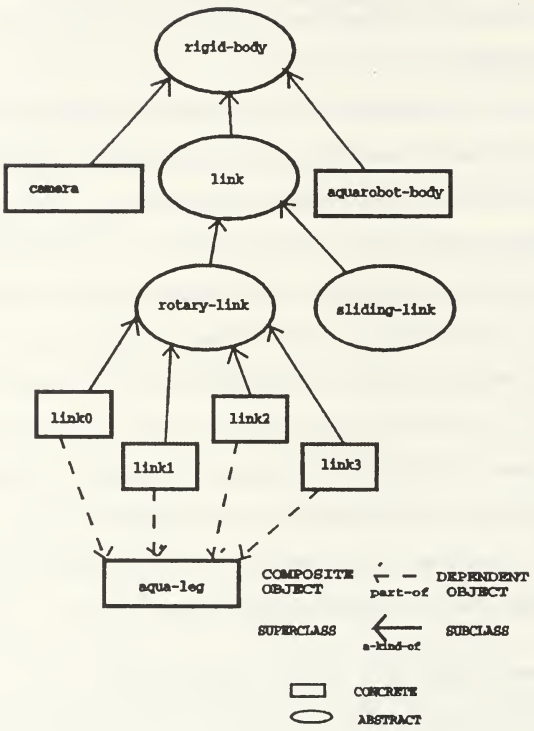


Figure 7.1

CLOS Aquarobot Class Diagram

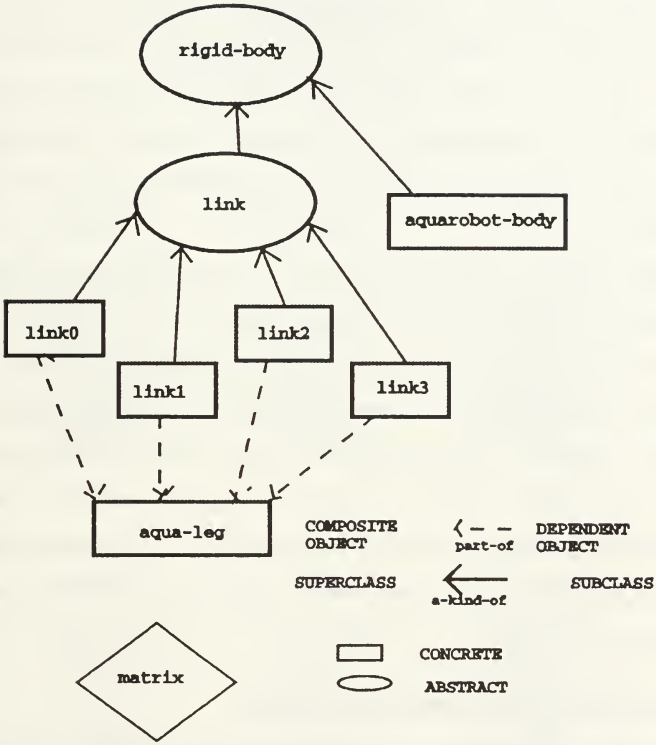


Figure 7.2

C++ Aquarobot Class Diagram

subclasses (link0 through link3) and a few numerical slots to create another top level class. The AquaLeg class also possesses functions that manipulate an AquaLeg type. The Matrix class is unique to the C++ version. The CLOS version uses lists to store data while the C++ version uses this defined Matrix class and its functions to store and manipulate the data. The Matrix class is a typical example of a class utility.

The object hierarchy used to instantiate an Aquarobot differs in the two versions. Figure 7.3, the C++ object diagram, constructs Aquarobot as seven subobjects which can be deleted or reproduced without affecting the existence of the other. Figure 7.4 displays the CLOS object diagram that has one top level object with dependent object hierarchy containing a total of thirty-one objects. The Leg object and its dependent subobjects are identical in each language version.

C. AQUAROBOT CLASS DEFINITION CODE

Classes are defined in various ways dependent upon the OO language used. There are, however, many similarities in class attributes. For example, both CLOS and C++ have slots for the items within a class. The AquaLeg class definition in both language versions is explained in the following paragraphs.

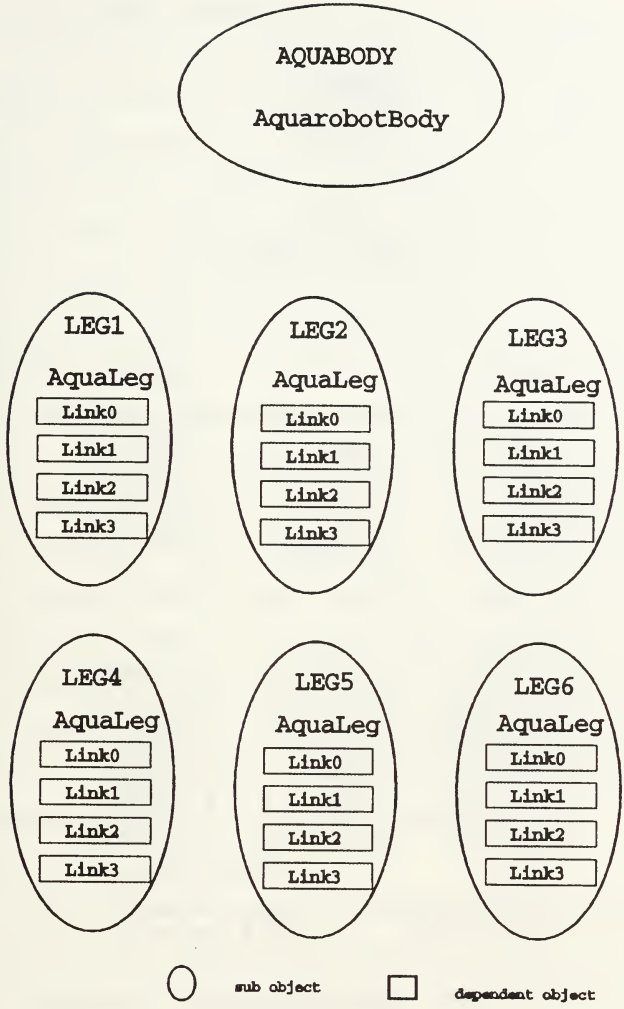


Figure 7.3

C++ Object Hierarchy

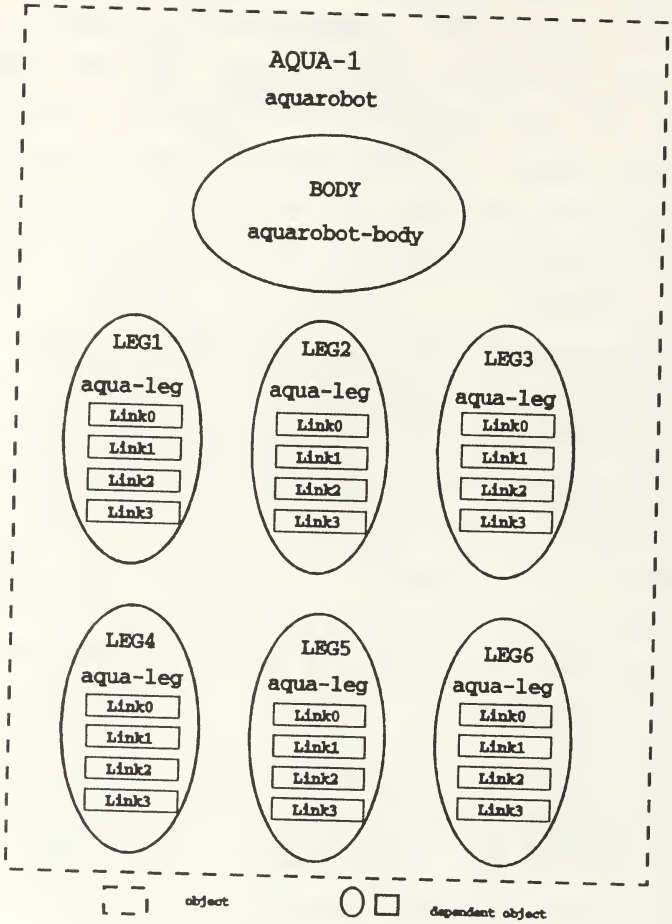


Figure 7.4

CLOS Object Hierarchy

1. CLOS Class Description

CLOS provides a template that contains both optional and mandatory information requirements. This template can be found in CLOS manuals. Figure 7.5 is the aqua-leg class defined using CLOS. Eight slots are defined and then initialized using the :initarg or :initform command. The dependent objects shown in Figure 7.4 are instantiated within the aqua-leg class as link0 through link3 using the make-instance command. The link0 class, for example, incorporates the superclass slots of Link.

The functions related to the aqua-leg class are defined outside of the class definition in defmethods. "Initialize-leg" is a function which requires an "aqua-leg" and an "aquarobot-body" class as input. Each input is given a local variable name of "leg" and "body" respectively. The functions may call other functions or change slot values. The CLOS code includes a camera class because the code was developed on a Sun workstation while the C++ version uses the graphics library on a Iris workstation.

2. C++ Class DESCRIPTION

The C++ AquaLeg class is defined within the AquaLeg.H file (Figure 7.6). Like the CLOS version in Figure 7.5, there are four dependent objects that are slots of the AquaLeg class. Like the CLOS version, functions which are applicable

```
(defclass aqua-leg ()
  ((leg-attachment-angle
    :initarg :leg-attachment-angle
    :accessor leg-attachment-angle)
   (link0
    :initform (make-instance 'link0)
    :accessor link0)
   (link1
    :initform (make-instance 'link1)
    :accessor link1)
   (link2
    :initform (make-instance 'link2)
    :accessor link2)
   (link3
    :initform (make-instance 'link3)
    :accessor link3)
   (motion-complete-flag
    :initform nil
    :accessor motion-complete-flag)
   (previous-foot-position
    :initform nil
    :accessor previous-foot-position)
   (current-foot-position
    :initform nil
    :accessor current-foot-position)))

(defmethod initialize-leg ((leg aqua-leg) (body aquarobot-body))
  (setf (inboard-link (link0 leg)) body)
  (setf (inboard-link (link1 leg)) (link0 leg))
  (setf (inboard-link (link2 leg)) (link1 leg))
  (setf (inboard-link (link3 leg)) (link2 leg))
  (rotate-link (link0 leg) (leg-attachment-angle leg))
  (rotate-link (link1 leg) (inboard-joint-angle (link1 leg)))
  (rotate-link (link2 leg) (inboard-joint-angle (link2 leg)))
  (rotate-link (link3 leg) (inboard-joint-angle (link3 leg)))
  (setf (current-foot-position leg)
        (ncar 3 (first (transformed-node-list (link3 leg))))))
```

Figure 7.5

CLOS Code Excerpt Defining and Implementating

Aquarobot Leg Kinematics

```
class AquaLeg
{
 public:

 // these dependent objects are instantiated
 Link0 *link0;
 Link1 *link1;
 Link2 *link2;
 Link3 *link3;

 // the flag is set to 1 if the motion is completed without
 // reaching any link limits
 int motion_complete_flag;

 // the flag is set to 1 if the leg is on the ground
 int leg_support_flag;

 // the angle off of leg one where the leg is attachec to
 // the body
 double leg_attachment_angle;

 AquaLeg(AquarobotBody&, double); // constructor and initializer
 ~AquaLeg();                      // destructor
 void MoveIncremental(AquarobotBody &, double delta1, double delta2,
      double delta3);
 double GetLegAttachmentAngle() { return leg_attachment_angle;}
 int GetMotionCompleteFlag() { return motion_complete_flag;}
 void SetLegAttachmentAngle(double angle) {leg_attachment_angle = angle;}
 void SetMotionCompleteFlag(int flag) {motion_complete_flag = flag;}
 int GetLegSupportFlag() { return leg_support_flag;}
 void SetLegSupportFlag(int flag) {leg_support_flag = flag;}
};
#endif
```

Figure 7.6

C++ Code Excerpt Defining Aquarobot

Leg Kinematics

to the AquaLeg class are included within the class definition. An example, shown in that figure, is the "Move Incremental" function which increments the joint angles of a specified leg by a given amount.

The C++ function similar to the CLOS "initialize-leg" function is the C++ constructor "AquaLeg" shown in Figure 7.6. This function requires an AquarobotBody class and a double number as inputs. This and the other AquaLeg functions are defined within the AquaLeg.C file (Figure 7.7). Like the CLOS version, it is within the constructor that the dependent objects, Link0 through Link3, are instantiated. Similar to the CLOS version, other functions may be called or slot values altered. The "matrix" class, found within the MatrixMy.C and MatrixMy.H files (see Appendix C), is a class utility and is used within the AquaLeg constructor.

D. AQUAROBOT OBJECT INSTANTIATION CODE

Objects may be constructed in various composition within an OOP. However, the method of actually instantiating an object varies among OO languages. The CLOS version, shown in Figure 7.4, displays one top level object while the C++ program, shown in Figure 7.3, makes seven subobjects to produce one Aquarobot system. This section will discuss the individual language's method of instantiation using the two Aquarobot versions.

```
// ************************************************************
// FUNCTION: ~AquaLeg()
// PURPOSE: destructor of AquaLeg class
// ************************************************************

AquaLeg::~AquaLeg()
{
  delete link0;
  delete link1;
  delete link2;
  delete link3;
}

// ************************************************************
// FILENAME: AquaLeg
// PURPOSE:  constructor of AquaLeg class
// RETURNS:  AquaLeg class with values
// ************************************************************

AquaLeg::AquaLeg(AquarobotBody &body, double angle)
{
  motion_complete_flag = 1;     // initializes flag value
  SetLegAttachmentAngle(angle);
  link0 = new Link0;
  link1 = new Link1;
  link2 = new Link2;
  link3 = new Link3;


// initial link values initialized

// temp matrix adds in the T_matrix needed for the physical
// attachment of the leg to the body
   matrix temp;

// updates the Transformation matrix from body center to the
// leg attachment point
   temp.UpdateTMatrix(GetLegAttachmentAngle(),0.,0.,0.);
   temp = *body.H_matrix * temp;
   link0->RotateLink(&temp ,link0->GetInboardJointAngle());

   link1->RotateLink(link0->H_matrix, link1->GetInboardJointAngle());
   link2->RotateLink(link1->H_matrix,link2->GetInboardJointAngle());
   link3->RotateLink(link2->H_matrix,link3->GetInboardJointAngle());
}
```

Figure 7.7

C++ Code Excerpt Implementing Aquarobot

Leg Kinematics

1. CLOS Object Description

The CLOS version produces an Aquarobot by performing the function "aqua-picture" in the LISP screen environment. This function's code is displayed in Figure 7.8 and instantiates a top level object, Aquarobot, (named "aqua-1") in its first line using the make-instance command. The class "aquarobot" is used as the blueprint for this instantiation. This class consists of one body and six legs ("leg1" through "leg6") as dependent objects. These slots are instantiated using the same make-instance command when an "aquarobot" is created. Slot values are instantiated within the aqua-leg instantiation using the variable :leg-attachment-angle which was the initializing argument for a slot with the same name in the aqua-leg class (Figure 7.5).

2. C++ Object Description

Bot.C (Figure 7.9) is the main program the C++ version. This program controls the construction of the Aquarobot. The AquarobotBody and six AquaLegs are instantiated at this top level and they are all subobjects since there is no explicit Aquarobot instantiated. Similar to CLOS, each object is given a name (for example "leg3") and initialization values at the same time it is instantiated. C++ does not provide a command that equates to CLOS's make-

```
(defclass aquarobot ()
  ((body
    :initform (make-instance 'aquarobot-body)
    :accessor body)
   (leg1
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 0))
    :accessor leg1)
   (leg2
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 60))
    :accessor leg2)
   (leg3
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 120))
    :accessor leg3)
   (leg4
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 180))
    :accessor leg4)
   (leg5
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 240))
    :accessor leg5)
   (leg6
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 300))
    :accessor leg6)))

(defmethod initialize ((aqua aquarobot))
  (transform-node-list (body aqua))
  (initialize-leg (leg1 aqua) (body aqua))
  (initialize-leg (leg2 aqua) (body aqua))
  (initialize-leg (leg3 aqua) (body aqua))
  (initialize-leg (leg4 aqua) (body aqua))
  (initialize-leg (leg5 aqua) (body aqua))
  (initialize-leg (leg6 aqua) (body aqua)))

(defun aqua-picture ()
  (setf aqua-1 (make-instance 'aquarobot))
  (initialize aqua-1)
  (setf camera-1 (make-instance 'camera))
  (create-camera-window camera-1)
  (take-picture camera-1 aqua-1))
```

Figure 7.8

CLOS Code For Aquarobot Class

```
main()
{

  /* value returned from the event queue */

  short value;
  long  mainmenu;

  long  hititem;

  FILE *ifp;
  ifp = fopen("bot.dat","r");

  /* initialize the IRIS system */
  initialize();

   /* Create Pop Up Menus */
  mainmenu = makethemenus();

  // make the robot from its pieces
  AquarobotBody aquabody;
  AquaLeg leg1(aquabody,0.0);
  AquaLeg leg2(aquabody,60.0);
  AquaLeg leg3(aquabody,120.0);
  AquaLeg leg4(aquabody,180.0);
  AquaLeg leg5(aquabody,240.0);
  AquaLeg leg6(aquabody,300.0);
```

Figure 7.9

C++ Code Excerpt From Main Program Showing

Instantiation of the Parts of Aquarobot

instance command. Instead, C++ instantiates an object by declaring the class and providing a name and information necessary.

E. GRAPHICS

1. Graphics Display

The CLOS version of Aquarobot (Appendix A) was graphically simulated on a low end Silicon Graphics Indigo graphics workstation using a LISP camera object. This is the camera.cl file in Appendix A and it was created as a debugging tool because CLOS does not provide a graphics capability within its library. Examples of the CLOS graphics produced by a camera object are in Appendix C. The C++ version of Aquarobot (Appendix B) was simulated on the same workstation as the CLOS code. The C++ code uses the system's basic graphic library, gl. The C++ graphics code will be discussed in this section.

The C++ simulation was developed to support the debugging of control software. A user's manual explaining its use in this application has been produced (Suzuki, 1993). The graphics code is included in bot.C in Appendix B. This file includes the main program (a requirement of C++) which provides the initial instantiation of Aquarobot and controls function calls. This file also provides the graphics setup (in the initialize function) and the function that draws the

stick figure Aquarobot (in the drawaqua function). This coordination of bot.C is depicted in the flow diagram in Figure 7.10.

Aquarobot is instantiated in the reset position. Figure 7.11 displays this first graphical view. The next motion for Aquarobot is provided from the output of the gait algorithm. The information is provided by the position and orientation change of the body and the change in joint angle for each joint of each leg. The respective changes are transferred to the body's MoveIncremental function and leg MoveIncremental function. These functions update desired body and link positions. The body's function is called first because each leg uses the body's updated H matrix computation in their functions. The function FindPositions determines the Cartesian coordinate location of each joint and the footpad for each leg, as well as the body's position and orientation. Figure 7.12 shows a change ordered in one of leg one's joint angles. The C++ graphics code continuously polls for an acceptable queued signal. This signal determines the path taken and the functions performed within that option.

The CLOS version uses the user interface on the terminal as its main program. As shown in the script file of Appendix C, this method does not use an explicit continuous polling loop like C++. Rather, LISP provides an infinite read-eval-print loop within its user interface. Appendix C

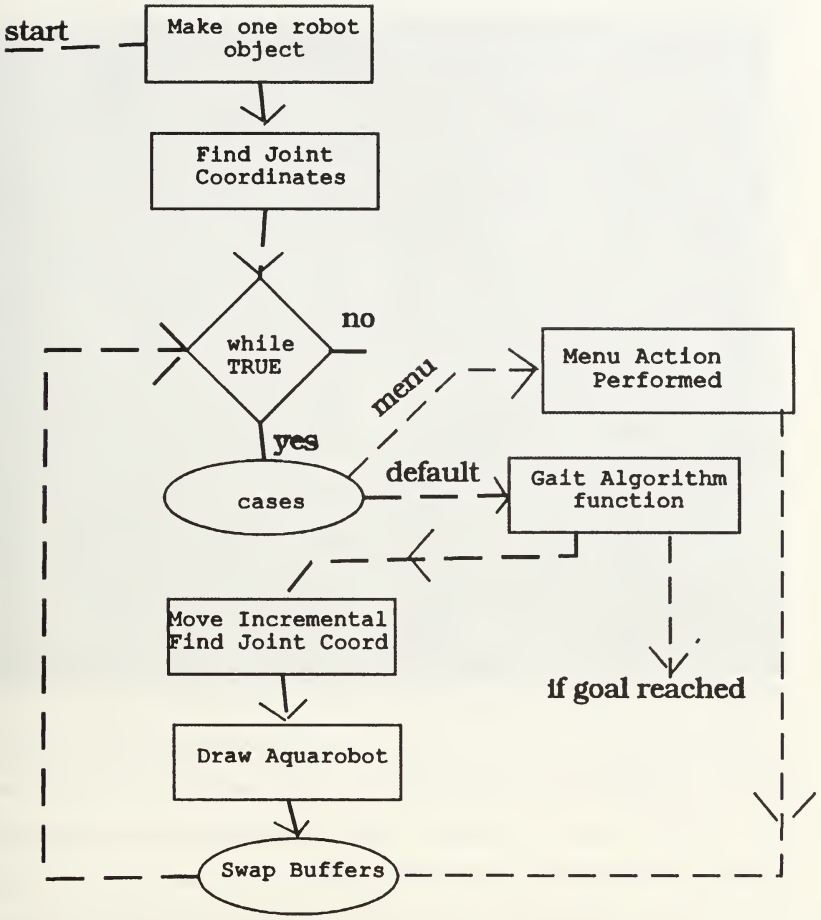


Figure 7.10

C++ Program Flow Diagram for Main Program

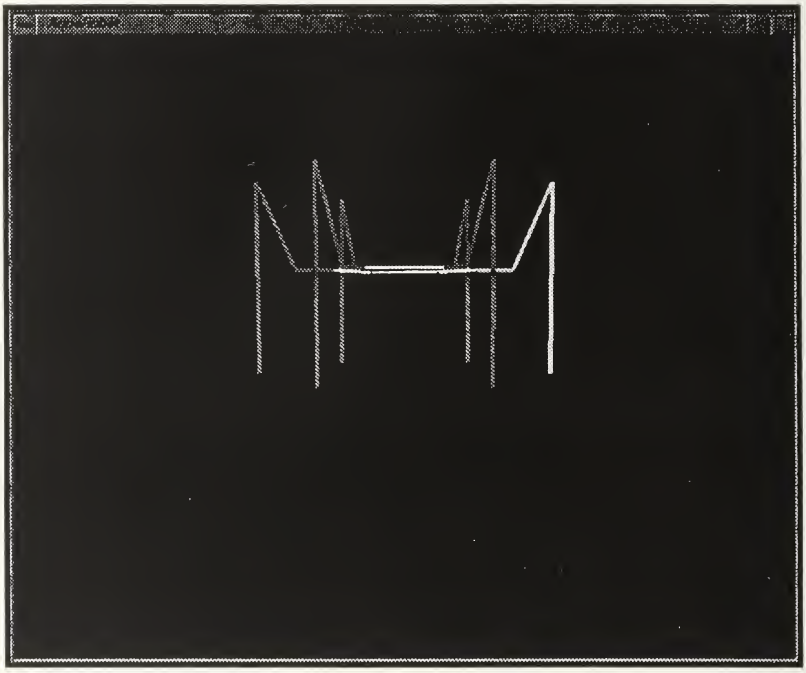


Figure 7.11

Initial Aquarobot Graphic Display

also provides examples of Aquarobot graphics obtained using the code in Appendix A.

2. User Interface

In the C++ version of the Aquarobot simulation code, one acceptable queue signal is the clicking of an option on the menu shown upon the screen. Figure 7.12 displays the menu and its options. The options provide various camera views of Aquarobot and the ability to read from a data file that consists of data changes. The camera views are particularly helpful in the debugging of gait motions conducted in the gait algorithm function.

F. SUMMARY

C++ and CLOS are both similar in their method of defining a class. The instantiation technique differs between the languages with C++ requiring a class constructor function that defines the instantiation while CLOS uses a reserved command and the class definition.

The Aquarobot programs were not designed from identical class and object hierarchies and this provides varied examples of design as well as OO language variations. Appendices A and B include the entire CLOS and C++ codes. Appendix C provides a script file and graphical pictures produced using the CLOS version. The next chapter evaluates the CLOS and C++ codes and their performance.

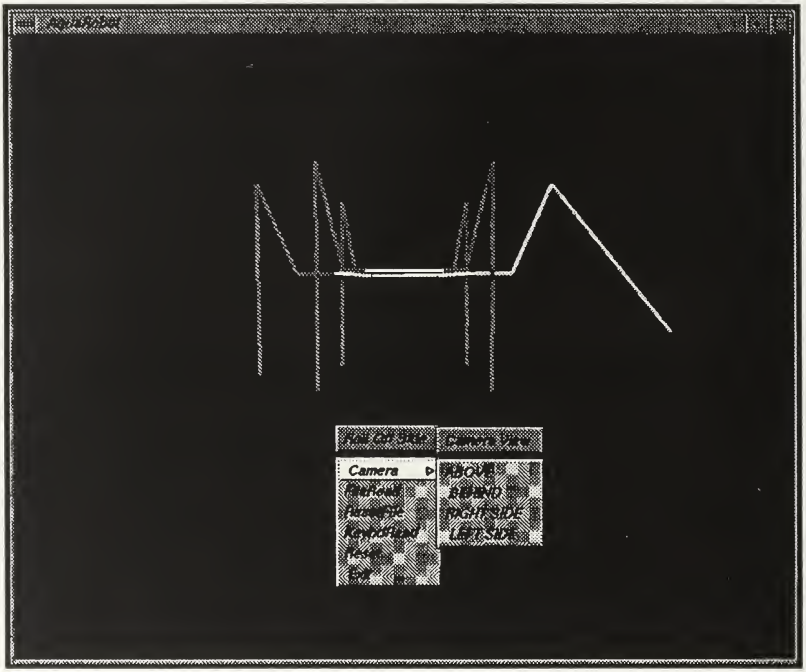


Figure 7.12

Ordered Motion Aquarobot Graphic Display

VIII. EVALUATION

A. INTRODUCTION

Both the CLOS and C++ versions were successful in producing a graphic simulator of Aquarobot. Each code, however, varies in its graphical performance and code characteristics.

B. CLOS/C++ CODE EVALUATION

The CLOS code usually requires less lines of code to write a function than C++. The codes in Appendices A and B display approximately a three to one ratio of length in favor of CLOS. This compact code provides CLOS an advantage in understandability and prototyping. Unfortunately, CLOS function definitions, when optimized for conciseness, may be rather cryptic. C++, although longer in length, is similar in format to many more common languages, and therefore it is in some ways easier to read, especially by programmers not skilled in CLOS and LISP.

CLOS provides dynamic memory allocation and uses garbage collection to accumulate unused memory space. Activity is suspended during garbage collection methods. In contrast, C++ uses a memory heap which requires memory to be removed and then explicitly returned to the heap when the memory space is no longer needed.

C++ and CLOS are both similar in their method of defining a class. The instantiation technique differs between the languages with C++ requiring a class constructor function that defines the instantiation while CLOS uses a reserved command, make-instance, and the class title.

C. CLOS/C++ GRAPHICAL EVALUATION

Both Aquarobot versions were simulated with the same leg and body motions. The time required to calculate new joint parameters via kinematics was recorded. The compiled CLOS version requires 310 ms for execution and display of one move (six degrees of freedom for body motion and 18 leg joint motions) while 160 ms is required for the same result using the C++ version.

The numerical results above were run without either code being optimized, nor were compiler switches set for optimized code generations. The C++ version performed faster than the CLOS version by a factor of two to one. The C++ speed advantage was not noteworthy enough to recommend against using CLOS unless execution efficiency is the dominant factor. In addition, CLOS required less programming effort (i.e., more compact code) than C++. This author recommends CLOS as a prototype executable specification code when a complex systems are involved. However, the common format and familiarity of C++ provides a better comprehension of the code by others who

might use or modify it (which are major advantages). In addition, it appears that C++'s two to one speed ratio may be improved with optimization.

D. SUMMARY

The C++ and CLOS Aquarobot objects have been designed using classes that are common to all rigid body manipulators. The generic classes are defined in both code versions and can be incorporated into other robot design code. This reusability of OO code as well as its simple modification steps are advantages to this design in this and other research projects, since future researchers may build on current work. This author's C++ code is currently being used in Aquarobot dynamics research for a dissertation at Ohio State University (McMillan, 1992).

The C++ code is also being used for its original purpose as a simulator to debug control algorithms. Specifically, Master degree students at the Naval Postgraduate School are currently using this simulator to investigate gait algorithms (Schue, 1993). The graphical simulator provides a useful and time saving method to check this work visually. It also provides an environment where innovative methods for motion can be tried without requiring the physical robot to be put into danger. The last chapter of this thesis provides conclusions concerning this research project and suggests topics for future research.

IX. CONCLUSIONS

A. INTRODUCTION

This thesis represents the beginning of a major research effort to provide a generic walking robot simulator, although the concepts developed here can be used for any articulated rigid body system The overall research project is centered around Aquarobot, a six-legged walking machine developed in Japan. While the techniques and the computer programs developed and used here are only a small part of the overall effort of the Naval Postgraduate School Aquarobot project, they represent critical first steps.

B. FUTURE USE OF CODE IN OTHER ROBOT DESIGNS

The codes discussed in this thesis and included in entirety in Appendices A and B provide the basic classes necessary to create any rotary link manipulator objects. The C++ code (Appendix B) provides the core structure that any design requires to produce a graphic simulator using Silicon Graphics Iris systems. The CLOS version (Appendix A) includes a camera class that can be used on any SUN workstation as well as any Iris system which has been provided with a LISP compiler.

C. FUTURE USE OF AQUAROBOT

Aquarobot's underwater walking capability provide many possibilities for its future use. Aside from its original application to assist with the quality control and construction supervision for tsunamai sea wall foundations, other possibilities include its use at marinas to locate large items dropped into the water, and at lakes or shallow beaches when an underwater search is needed. Aquarobot could also be helpful in detecting underwater cracks in piping such as electric cables and gas lines.

Additionally, the Aquarobot concept presents many possible uses in military applications. For example, an aquatic walking machine provides an alternative method of mine detection on the sea floor or in a surf zone. Upon completion of an analysis of the feasibility of walking between land and water, it may be found that this class of robots could be used for beach inspection prior to an amphibious landing, thus saving lives.

D. FUTURE RESEARCH IDEAS

This thesis describes the initial stage of the Aquarobot research project. There are a number of avenues of future reasearch. Some of these areas include: simulating the robot's dynamics, modeling the joint motors, modeling Aquarobot's hinged foot pads, modeling the hydrodynamic

effects of legged walking machines, improving the graphic portion, and gait planning research.

The graphics area alone provides a number of areas of research such as: providing collision detection and creating an Aquarobot replica that is more faithful in appearance to the actual robot. Both graphics codes currently display a stick figure.

Aside from this thesis' use of CLOS as executable specification code for a C++ final version, CLOS could also be used in incremental development of C++. This would take advantage of CLOS's superior debugging environment. Another alternative in code production would be to use CLOS as the main program and import C++ functions. This may in the end prove to be the best way to construct an interactive simulator, but this has yet to be investigated.

E. SUMMARY

Robotics engineers have made impressive progress in accomplishing the goal of producing an effective and practical walking machine. Such a machine will permit society to achieve functions and carry out missions not presently possible. Also, they will improve current methods of performing tasks we do now but not very efficiently, or with considerable danger to human workers.

The legged robot concept is in its infancy. It is vital that we have simulation tools capable of providing an

accurate model of complicated linkage mechanisms used for manipulation and locomotion. These tools reduce the need for a prototype vehicle in the early stages of development, and provide a safe environment to attempt unusual and possibly dangerous tests. Trying untested gaits, designs, etc. in a simulation environment allows for better understanding of the performance envelope and capabilities of a walking machine, saves money, and potentially saves lives.

Simulation tools such as modeling algorithms, powerful computer languages, and graphical capabilities enhance and promote technology. There are many ways to build a simulation model, but the use of the combination of kinematic modeling, object oriented languages, and graphically equipped computer systems offers a flexible and robust design method for both large complex robots and simple one limb imitations of organisms.

Aquarobot is a six-legged walking robot whose software will be improved using this simulation model. Later generations of legged machines may also benefit from similar simulation studies. Aquarobot's actions and stability in various postures and with different gaits will be tested to enhance its efficiency and design. Hopefully, through these tests and improvements, legged machines will be able to exploit their many advantages in areas where wheeled vehicles are now used, as well as permitting access to areas where no vehicle can now go.

APPENDIX A - CLOS CODE

link.cl

```
(defclass link (rigid-body)
  ((motion-limit-flag
    :initform nil
    :accessor motion-limit-flag)
   (twist-angle
    :initarg :twist-angle
    :accessor twist-angle)
   (link-length
    :initarg :link-length
    :accessor link-length)
   (inboard-joint-angle
    :initarg :inboard-joint-angle
    :accessor inboard-joint-angle)
   (inboard-joint-displacement
    :initarg :inboard-joint-displacement
    :accessor inboard-joint-displacement)
   (inboard-link
    :initarg :inboard-link
    :accessor inboard-link)
   (A-matrix
    :accessor A-matrix)))

(defclass rotary-link (link)
  ((min-joint-angle
    :initarg :min-joint-angle
    :accessor min-joint-angle)
   (max-joint-angle
    :initarg :max-joint-angle
    :accessor max-joint-angle)))

(defclass sliding-link (link)
  ((min-joint-displacement
    :initarg :min-joint-displacement
    :accessor min-joint-displacement)
   (max-joint-displacement
    :initarg :max-joint-displacement
    :accessor max-joint-displacement)))
```

aqua-link.cl

```
(defclass link0 (rotary-link)
  ((twist-angle :initform 0)
   (link-length :initform 37.5)
   (inboard-joint-angle :initform 0)
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -360))
   (max-joint-angle :initform (deg-to-rad 360))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-37.5 0 0 1)))
   (polygon-list :initform '((1 2)))))

(defclass link1 (rotary-link)
  ((twist-angle :initform (deg-to-rad -90))
   (link-length :initform 20)
   (inboard-joint-angle :initform 0)
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -60))
   (max-joint-angle :initform (deg-to-rad 60))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-20 0 0 1)))
   (polygon-list :initform '((1 2)))))

(defclass link2 (rotary-link)
  ((twist-angle :initform 0)
   (link-length :initform 50)
   (inboard-joint-angle :initform (deg-to-rad 66.4))
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -106.6))
   (max-joint-angle :initform (deg-to-rad 73.4))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-50 0 0 1)))
   (polygon-list :initform '((1 2)))))

(defclass link3 (rotary-link)
  ((twist-angle :initform 0)
   (link-length :initform 100)
   (inboard-joint-angle :initform (deg-to-rad -156.4))
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -156.4))
   (max-joint-angle :initform (deg-to-rad 23.6))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-100 0 0 1)))
   (polygon-list :initform '((1 2)))))

(defmethod update-A-matrix ((link link))
  (with-slots (twist-angle link-length inboard-joint-angle
     inboard-joint-displacement A-matrix) link
     (setf A-matrix (dh-matrix (cos inboard-joint-angle)
       (sin inboard-joint-angle) (cos twist-angle) (sin twist-angle)
       link-length inboard-joint-displacement))))

(defun deg-to-rad (angle) (* .0174532925199433295 angle))

(defmethod rotate ((link rotary-link) angle)
  (setf (inboard-joint-angle link) angle)
  (update-A-matrix link)
  (setf (H-matrix link) (matrix-multiply (H-matrix (inboard-link link))
                                         (A-matrix link)))
  (transform-node-list link))

(defmethod rotate-link ((link rotary-link) angle)
  (cond ((> angle (max-joint-angle link))
                                (rotate link (max-joint-angle link))
                                (setf (motion-limit-flag link) t))
        ((< angle (min-joint-angle link))
                                (rotate link (min-joint-angle link))
                                (setf (motion-limit-flag link) t))
        (t (rotate link angle) (setf (motion-limit-flag link) nil))))
```

```
(defclass aqua-leg ()
  ((leg-attachment-angle
    :initarg :leg-attachment-angle
    :accessor leg-attachment-angle)
   (link0
    :initform (make-instance 'link0)
    :accessor link0)
   (link1
    :initform (make-instance 'link1)
    :accessor link1)
   (link2
    :initform (make-instance 'link2)
    :accessor link2)
   (link3
    :initform (make-instance 'link3)
    :accessor link3)
   (motion-complete-flag
    :initform nil
    :accessor motion-complete-flag)
   (previous-foot-position
    :initform nil
    :accessor previous-foot-position)
   (current-foot-position
    :initform nil
    :accessor current-foot-position)))

(defmethod initialize-leg ((leg aqua-leg) (body aquarobot-body))
  (setf (inboard-link (link0 leg)) body)
  (setf (inboard-link (link1 leg)) (link0 leg))
  (setf (inboard-link (link2 leg)) (link1 leg))
  (setf (inboard-link (link3 leg)) (link2 leg))
  (rotate-link (link0 leg) (leg-attachment-angle leg))
  (rotate-link (link1 leg) (inboard-joint-angle (link1 leg)))
  (rotate-link (link2 leg) (inboard-joint-angle (link2 leg)))
  (rotate-link (link3 leg) (inboard-joint-angle (link3 leg)))
  (setf (current-foot-position leg)
        (ncar 3 (first (transformed-node-list (link3 leg))))))

(defmethod take-picture ((camera camera) (leg aqua-leg))
  (take-picture camera (link1 leg))
  (take-picture camera (link2 leg))
  (take-picture camera (link3 leg)))

(defmethod move-incremental ((leg aqua-leg) increment-list)
  (rotate-link (link0 leg) (leg-attachment-angle leg))
  (rotate-link (link1 leg)
               (+ (first increment-list) (inboard-joint-angle (link1 leg))))
  (rotate-link (link2 leg)
               (+ (second increment-list) (inboard-joint-angle (link2 leg))))
  (rotate-link (link3 leg)
               (+ (third increment-list) (inboard-joint-angle (link3 leg))))
  (setf (previous-foot-position leg) (current-foot-position leg))
  (setf (current-foot-position leg)
        (ncar 3 (first (transformed-node-list (link3 leg)))))
  (setf (motion-complete-flag leg) (not (or (motion-limit-flag (link1 leg))
          (motion-limit-flag (link2 leg)) (motion-limit-flag (link3 leg))))))

(defmethod feasible-movep ((leg aqua-leg) allowable-sinkage allowable-slippage)
  (and (<= (third (current-foot-position leg)) allowable-sinkage)
       (or (minusp (third (current-foot-position leg)))
           (minusp (third (previous-foot-position leg)))
           (<= (vector-length (vector-slippage leg)) allowable-slippage))))

(defmethod vector-slippage ((leg aqua-leg))

 (vector-subtract (rest (reverse (previous-foot-position leg)))
                  (rest (reverse (current-foot-position leg)))))
```

```
(defclass aquarobot-body (rigid-body)
   ((node-list
     :initform '((0 0 0 1) (37.5 0 0 1) (18.75 32.48 0 1)
                 (-18.75 32.48 0 1) (-37.5 0 0 1) (-18.75 -32.48 0 1)
                 (18.75 -32.48 0 1) (37.5 0 -15 1)))
    (polygon-list
     :initform '((1 2 3 4 5 6) (1 7)))
    (H-matrix
     :initform (homogeneous-transform 0 0 0 0 0 z-init))))

(defclass aquarobot ()
  ((body
    :initform (make-instance 'aquarobot-body)
    :accessor body)
   (leg1
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 0))
    :accessor leg1)
   (leg2
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 60))
    :accessor leg2)
   (leg3
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 120))
    :accessor leg3)
   (leg4
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 180))
    :accessor leg4)
   (leg5
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 240))
    :accessor leg5)
   (leg6
    :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 300))
    :accessor leg6)))

(defmethod initialize ((aqua aquarobot))
  (transform-node-list (body aqua))
  (initialize-leg (leg1 aqua) (body aqua))
  (initialize-leg (leg2 aqua) (body aqua))
  (initialize-leg (leg3 aqua) (body aqua))
  (initialize-leg (leg4 aqua) (body aqua))
  (initialize-leg (leg5 aqua) (body aqua))
  (initialize-leg (leg6 aqua) (body aqua)))

(defun aqua-picture ()
  (setf aqua-1 (make-instance 'aquarobot))
  (initialize aqua-1)
  (setf camera-1 (make-instance 'camera))
  (create-camera-window camera-1)
  (take-picture camera-1 aqua-1))

(defmethod take-picture ((camera camera) (aqua aquarobot))
  (take-picture camera (body aqua))
  (take-picture camera (leg1 aqua))
  (take-picture camera (leg2 aqua))
  (take-picture camera (leg3 aqua))
  (take-picture camera (leg4 aqua))
  (take-picture camera (leg5 aqua))
  (take-picture camera (leg6 aqua)))

(defun new-picture ()
  (take-picture camera-1 aqua-1))

(defconstant z-init -54.181866)

(defmethod move-incremental ((aqua aquarobot) increment-list)
```

```
  (move-incremental (body aqua) (first increment-list))
  (move-incremental (leg1 aqua) (second increment-list))
  (move-incremental (leg2 aqua) (third increment-list))
  (move-incremental (leg3 aqua) (fourth increment-list))
  (move-incremental (leg4 aqua) (fifth increment-list))
  (move-incremental (leg5 aqua) (sixth increment-list))
  (move-incremental (leg6 aqua) (seventh increment-list)))

(defconstant null-move-list '((0 0 0 0 0 0) (0 0 0) (0 0 0) (0 0 0)
 (0 0 0) (0 0 0) (0 0 0)))

(defmethod feasible-movep ((aqua aquarobot) allowable-sinkage
                           allowable-slippage)
(and (feasible-movep (leg1 aqua) allowable-sinkage allowable-slippage)
     (feasible-movep (leg2 aqua) allowable-sinkage allowable-slippage)
     (feasible-movep (leg3 aqua) allowable-sinkage allowable-slippage)
     (feasible-movep (leg4 aqua) allowable-sinkage allowable-slippage)
     (feasible-movep (leg5 aqua) allowable-sinkage allowable-slippage)
     (feasible-movep (leg6 aqua) allowable-sinkage allowable-slippage)))
```

```
(require :xcw)
(cw:initialize-common-windows)

(defclass camera (rigid-body)
  ((focal-length
    :accessor focal-length
    :initform 6)
   (previous-point
    :accessor previous-point)
   (camera-window
    :accessor camera-window)
   (H-matrix
    :initform (homogeneous-transform .3 -.3 0 -300 -90 -90))
   (inverse-H-matrix
    :accessor inverse-H-matrix
    :initform (inverse-H (homogeneous-transform .3 -.3 0 -300 -90 -90)))
   (enlargement-factor
    :accessor enlargement-factor
    :initform 30)))

(defmethod create-camera-window ((camera camera))
  (setf (camera-window camera)
        (cw:make-window-stream :borders 5
                               :left 500
                               :bottom 500
                               :width 300
                               :height 300
                               :title "aquarobot"
                               :activate-p t)))

(defmethod move ((camera camera) azimuth elevation roll x y z)
  (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
  (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera))))

(defmethod take-picture ((camera camera) (body rigid-body))
  (dolist (polygon (polygon-list body))
          (draw-polygon camera polygon (transformed-node-list body))))

(defmethod draw-polygon ((camera camera) polygon node-coord-list)
  (let* ((starting-node-index (first polygon))
         (remaining-node-indices (rest polygon))
         (start-point-coord (nth starting-node-index node-coord-list)))
    (transform-and-move-pen-to camera start-point-coord)
    (dolist (node-index remaining-node-indices)
            (transform-and-draw-to camera (nth node-index node-coord-list)))
    (transform-and-draw-to camera start-point-coord)))  ;closes polygon

(defmethod transform-and-move-pen-to ((camera camera) point-in-earth-space)
  (setf (previous-point camera)
        (compute-camera-window-coordinates camera point-in-earth-space)))

(defmethod transform-and-draw-to ((camera  camera) point-in-earth-space)
  (let ((to-point
         (compute-camera-window-coordinates camera point-in-earth-space)))
    (draw-line-in-camera-window camera (previous-point camera) to-point)
    (setf (previous-point camera) to-point)))

(defmethod draw-line-in-camera-window ((camera camera) start end)
  (cw:draw-line (camera-window camera)
                (cw:make-position :x (first start) :y (second start))
                (cw:make-position :x (first end) :y (second end))
                :brush-width 0))

(defmethod compute-camera-window-coordinates ((camera camera)
```

```
                                                    point-in-earth-space)
  (let* ((enlargement-factor (enlargement-factor camera))
         (focal-length (focal-length camera))
         (point-in-camera-space (post-multiply (inverse-H-matrix camera)
                                               point-in-earth-space))
         (x (first point-in-camera-space))  ;x axis is along optical axis
         (y (second point-in-camera-space)) ;y is out right side of camera
         (z (third point-in-camera-space))) ;z is out bottom of camera

    (if (>= x focal-length)             ; handles rear clipping
        (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
                 150)                   ;to right in camera window
          (+ 150 (round (* enlargement-factor (/ (* focal-length (- z)) x)))))
                                        ;up in camera window
      (list -1 -1))))
```

load-files.cl 1

```
(load "camera.cl")
(load "link.cl")
(load "rigid-body.cl")
(load "robot-kinematics.cl")
(load "aqua.cl")
(load "aqua-leg.cl")
(load "aqua-link.cl")
```

```
(defclass rigid-body
    ()
  ((location
    :initarg :location
    :accessor location)
   (velocity
    :initarg :velocity
    :accessor velocity)
   (acceleration
    :accessor acceleration)
   (forces-and-torques
    :accessor forces-and-torques)
   (moments-of-inertia
    :initarg :moments-of-inertia
    :accessor moments-of-inertia)
   (mass
    :initarg :mass
    :accessor mass)
   (node-list
    :initarg :node-list
    :accessor node-list)
   (polygon-list
    :initarg :polygon-list
    :accessor polygon-list)
   (transformed-node-list
    :accessor transformed-node-list)
   (H-matrix
    :accessor H-matrix)
   (current-time
    :accessor current-time)))

(defmethod move ((body rigid-body) azimuth elevation roll x y z)
  (setf (H-matrix body)
    (homogeneous-transform azimuth elevation roll x y z))
  (transform-node-list body)
  (update-position body))

(defmethod move-incremental ((body rigid-body) increment-list)
  (setf (H-matrix body)
    (matrix-multiply (H-matrix body) (homogeneous-transform
                                      (first increment-list)
                                      (second increment-list)
                                      (third increment-list)
                                      (fourth increment-list)
                                      (fifth increment-list)
                                      (sixth increment-list))))
  (transform-node-list body)
  (update-position body))


(defmethod get-delta-t ((body rigid-body))
  (let* ((new-time (get-internal-real-time))
         (delta-t (/ (- new-time (current-time body)) 1000)))
    (setf (current-time body) new-time)
    delta-t))

(defmethod start-timer ((body rigid-body))
  (setf (current-time body) (get-internal-real-time)))

(defmethod update-rigid-body ((body rigid-body))
  (let ((delta-t (get-delta-t body)))
    (update-acceleration body)
    (update-H-matrix body delta-t)
    (transform-node-list body)
```

```
    (update-position body)
    (update-velocity body delta-t)))

(defmethod update-acceleration ((body rigid-body))
  (setf (acceleration body) ;;(list u-dot v-dot w-dot p-dot q-dot r-dot)
    (multiple-value-bind
        (Fx Fy Fz L M N   u v w p q r   Ix Iy Iz)
        (values-list
         (append
          (forces-and-torques body) (velocity body) (moments-of-inertia body)))
      (list (+ (* v r) (* -1 w q) (/ Fx (mass body))
               (* *gravity* (first (third (H-matrix body)))))
            (+ (* w p) (* -1 u r) (/ Fy (mass body))
               (* *gravity* (second (third (H-matrix body)))))
            (+ (* u q) (* -1 v p) (/ Fz (mass body))
               (* *gravity* (third (third (H-matrix body)))))
            (/ (+ (* (- Iy Iz) q r) L) Ix)
            (/ (+ (* (- Iz Ix) r p) M) Iy)
            (/ (+ (* (- Ix Iy) p q) N) Iz)))))

(defmethod update-velocity ((body rigid-body) delta-t)
  (setf (velocity body)
    (vector-add (velocity body)
                (scalar-multiply delta-t (acceleration body)))))

(defmethod update-H-matrix ((body rigid-body) delta-t)
  (setf (H-matrix body)
    (matrix-multiply
     (H-matrix body) (homogeneous-transform
                      (* delta-t (sixth (velocity body)))
                      (* delta-t (fifth (velocity body)))
                      (* delta-t (fourth (velocity body)))
                      (* delta-t (first (velocity body)))
                      (* delta-t (second (velocity body)))
                      (* delta-t (third (velocity body)))))))

(defmethod transform-node-list ((body rigid-body))
  (setf (transformed-node-list body)
    (mapcar #'(lambda (node-location)
                (post-multiply (H-matrix body) node-location))
            (node-list body))))

(defmethod update-position ((body rigid-body))
  (setf (location body) (ncar 3 (first (transformed-node-list body)))))

(defmethod get-node-polygon-list ((body rigid-body))
  (list (transformed-node-list body) (polygon-list body)))

(defconstant *gravity* 32.2185)
```

```
(defun transpose (A)
  (cond ((null (cdr A)) (mapcar 'list (car A)))
        (t (mapcar 'cons (car A) (transpose (cdr A))))))

(defun dot-product (x y)          ;A vector is a list of numerical atoms.
  (apply '+ (mapcar '* x y)))     ;A matrix is a list of row lists.

(defun vector-length (x) (sqrt (dot-product x x)))

(defun post-multiply (M x)
  (cond ((null (cdr M)) (list (dot-product (car M) x)))
        (t (cons (dot-product (car M) x) (post-multiply (cdr M) x)))))

(defun pre-multiply (x M)
  (post-multiply (transpose M) x))

(defun matrix-multiply (A B)
  (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
        (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))

(defun chain-multiply (L)
  (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
        (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))

(defun cycle-left (L) (mapcar 'row-cycle-left L))

(defun row-cycle-left (R) (append (cdr R) (list (car R))))

(defun cycle-up (M) (append (cdr M) (list (car M))))

(defun unit-vector (one-column length)
  (do ((n length (1- n))
       (R nil (cons (cond ((= one-column n) 1) (t 0)) R)))
      ((zerop n) R)))

(defun unit-matrix (n)
  (do ((row-number n (1- row-number))
       (I nil (cons (unit-vector row-number n) I)))
      ((zerop row-number) I)))

(defun concat-matrix (A B)
  (cond ((null A) B)
        (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))

(defun augment (A) (concat-matrix A (unit-matrix (length A))))

(defun normalize-row (R) (scalar-multiply (/ 1.0 (car R)) R))

(defun scalar-multiply (a x)
  (cond ((null x) nil)
        (t (cons (* a (car x)) (scalar-multiply a (cdr x))))))

(defun solve-first-column (M)
  (do* ((L1 M (cdr L1))
        (L2 (normalize-row (car M)))
        (L3 (list L2) (cons (vector-add (car L1)
                                        (scalar-multiply (- (caar L1)) L2)) L3)))
       ((null (cdr L1)) (reverse L3))))

(defun vector-add (x y) (mapcar '+ x y))

(defun vector-subtract (x y) (mapcar '- x y))

(defun square-car (M)
```

```
  (do ((m (length M))
       (L1 M (cdr L1))
       (L2 nil (cons (ncar m (car L1)) L2)))
      ((null L1) (reverse L2))))

(defun ncdr (n L) (cond ((zerop n) L) (t (cdr (ncdr (1- n) L)))))

(defun ncar (n L) (cond ((zerop n) nil)
                        (t (cons (car L) (ncar (1- n) (cdr L))))))

(defun nmax-car-first (n L)
  (append (max-car-first (ncar n L)) (ncdr n L)))

(defun matrix-inverse (M)
  (do ((M1 (max-car-first (augment M))
           (cond ((null M1) nil)
                 (t (nmax-car-first n (cycle-left (cycle-up M1))))))
       (n (1- (length M)) (1- n)))
      ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (square-car M1))))
      (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))

(defun max-car-first (L)
  (cond ((null (cdr L)) L)
        (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L))))) L
               (append (max-car-first (cdr L)) (list (car L)))))))

(defun dh-matrix (cosrotate sinrotate costwist sintwist length translate)
  (list (list cosrotate (- (* costwist sinrotate))
              (* sintwist sinrotate) (* length cosrotate))
        (list sinrotate (* costwist cosrotate)
              (- (* sintwist cosrotate)) (* length sinrotate))
        (list 0. sintwist costwist translate) (list 0. 0. 0. 1.)))

(defun homogeneous-transform (azimuth elevation roll x y z)
  (rotation-and-translation (sin azimuth) (cos azimuth) (sin elevation)
  (cos elevation) (sin roll) (cos roll) x y z))

(defun rotation-and-translation (spsi cpsi sth cth sphi cphi x y z)
  (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
              (+ (* cpsi sth cphi) (* spsi sphi)) x)
        (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
              (- (* spsi sth cphi) (* cpsi sphi)) y)
        (list (- sth) (* cth sphi) (* cth cphi) z)
        (list 0. 0. 0. 1.)))

(defun inverse-H (H)
  (let* ((minus-P (list (- (fourth (first H)))
                        (- (fourth (second H)))
                        (- (fourth (third H)))))
         (inverse-R (transpose (square-car (reverse (rest (reverse H))))))
         (inverse-P (post-multiply inverse-R minus-P)))
        (append (concat-matrix inverse-R (transpose (list inverse-P)))
                (list (list 0 0 0 1)))))
```

APPENDIX B - C++ CODE

bot.h 1

```
// ********************************************************
//  FILENAME: bot.h
//  PURPOSE: defines constants and functions used int bot.C
//  NOTE: This is an IRIS 3D program written in C++
//  AUTHOR: S L Davidson
//  DATE: 5 January 1993
// ********************************************************

// provides constants for menu processing options
#define CAMERA   1
#define ABOVE    2
#define BEHIND   3
#define RTSIDE   4
#define LTSIDE   5
#define FILEREAD 6
#define KEYBDREAD 7
#define RESETFILE 8
#define RESET    14
#define EXIT     15


# define NEARCLIPPING 10.0  // planes defined
#define FARCLIPPING 1023.0

#define VIEWX = 0.0  /* location of the viewpoint */
#define VIEWY = 40.0
#define VIEWZ = 400.0

#define REFX = 0.0  /* location of the robot */
#define REFY = 0.0
#define REFZ = -200.0


long makethemenus();

static float rfx;/* reference point on in the x direction */
static float rfy;/* reference point on in the y direction */
static float rfz;/* reference point on in the z direction */

static float vx; /* view point on in the x direction */
static float vy; /* view point on in the y direction */
static float vz; /* view point on in the z direction */

double delta1,delta2,delta3;
double delaz,delel,delrol,delx,dely,delz;

void processmenuhit(long hititem);

void initialize(); // initializes graphics layout

void loadunit(); // a unit matrix used in rotation/translation

void projectionandviewingmatrix(float vx,float vy,float vz,float refx,float refy,float ref
z);

void buildnonmovingviewingmatrix(float vx,float vy,float vz,float refx,float refy,float re
fz);

void drawaqua(double*, double*, double *, double *, double *,
   double *, double *); // includes all legs and body
```

```
// ************************************************************
//  FILENAME: bot.C
//  PURPOSE: This file makes a stick aquarobot graphics
//                interactive design
//            It utilizes Kinematic functions to determine xyz
//                coordinates
//  CONTAINS: functions shown in bot.h
//  NOTE: This is and IRIS 3D program written in C++
//  AUTHOR: S L Davidson
//  DATE:  15 February 1993
// ************************************************************

#include "gl.h"  // graphics library
#include "device.h" // graphics library file
#include "bot.h"    // declaration file
#include <stdio.h>  // C++ library
#include "Link.H"
#include "RigidBody.H"
#include "MatrixMy.H"
#include "AquarobotBody.H"
#include "Kinematics.C"
#include "AquaLeg.H"

main()
{

// value returned from the event queue
 short value;
 long  mainmenu;
 long  hititem;

  FILE *ifp;
  ifp = fopen("bot.dat","r");

 // initialize the IRIS system
  initialize();

 // Create Pop Up Menus
  mainmenu = makethemenus();

 // makes the robot from its pieces
  AquarobotBody aquabody;
  AquaLeg leg1(aquabody,0.0);
  AquaLeg leg2(aquabody,60.0);
  AquaLeg leg3(aquabody,120.0);
  AquaLeg leg4(aquabody,180.0);
  AquaLeg leg5(aquabody,240.0);
  AquaLeg leg6(aquabody,300.0);

  Return_Coordinates coord;
  Passing_Items pass;
  Next_Motion trans;
  int ans;
  ans = 0;
  pass.legnum = 9;

  coord = FindPositions(aquabody, leg1, leg2, leg3, leg4, leg5, leg6);
  trans = TransferToGait(coord, aquabody);

  while(TRUE)
```

```
{

  // do we have something on the event queue?
   if(qtest())
    {

      switch(qread(&value))
       {
         case REDRAW:

           reshapeviewport();
           break;

       case MENUBUTTON:

           if (value == 1)
              [
               hititem = dopup(mainmenu);
               processmenuhit(hititem);
               ]
            break;

   //   queue used for calling the gait algorithm
        case AKEY:
             trans = Gait Algorithm(trans);

             delaz = trans.bodymotion[0];
             delel = trans.bodymotion[1];
             delrol = trans.bodymotion[2];
             delx = trans.bodymotion[3];
             dely = trans.bodymotion[4];
             delz = trans.bodymotion[5];

             aquabody.MoveIncremental(delaz,delel,delrol,delx,dely,delz);
             leg1.MoveIncremental(aquabody,trans.leg1motion[0],
trans.leg1motion[1],trans.leg1motion[2]);
             leg2.MoveIncremental(aquabody,trans.leg2motion[0],
trans.leg2motion[1],trans.leg2motion[2]);
             leg3.MoveIncremental(aquabody,trans.leg3motion[0],
trans.leg3motion[1],trans.leg3motion[2]);
             leg4.MoveIncremental(aquabody,trans.leg4motion[0],
trans.leg4motion[1],trans.leg4motion[2]);
             leg5.MoveIncremental(aquabody,trans.leg5motion[0],
trans.leg5motion[1],trans.leg5motion[2]);
             leg6.MoveIncremental(aquabody,trans.leg6motion[0],
trans.leg6motion[1],trans.leg6motion[2]);

           coord = FindPositions(aquabody,leg1,leg2,leg3,leg4,leg5,leg6);
           trans = TransferToGait(coord,aquabody);
           break;

       // reads incremental changes from a file
       case PKEY:
            printf("\n Reading file motion\n");
            pass = File_Use(ifp,aquabody,leg1,leg2,leg3,
                            leg4,leg5,leg6);
            printf("%d, %lf, %lf, %lf\n",pass.legnum,
                    pass.del1,pass.del2,pass.del3);
            pass.legnum = 9;

            coord = FindPositions(aquabody,leg1,leg2,leg3,
                                  leg4,leg5,leg6);
            trans = TransferToGait(coord,aquabody);
            break;
```

```
            case OKEY:
               rewind(ifp);
               break;

          default:

                break;

             }   // end switch on event queue item

          }   // endif qtest()


      color(BLUE);  // background color
      clear();

       // turn on Z-buffering
       zbuffer(TRUE);

       // clear the z-buffer
       zclear();

      buildnonmovingviewingmatrix(vx,vy,vz,rfx,rfy,rfz);
      drawaqua((coord.bodyc),(coord.leg1c),(coord.leg2c),(coord.leg3c),
               (coord.leg4c),(coord.leg5c),(coord.leg6c));

       // turn z-buffering off
       zbuffer(FALSE);

       // change the buffers
       swapbuffers();

       qenter(AKEY,1);
     }

}  // end of main




// ************************************************
// FUNCTION: INITIALIZE()
// ************************************************
void initialize()
{

  // set up the preferred aspect ratio
  keepaspect(XMAXSCREEN+1,YMAXSCREEN+1);

  // set up window size
  prefposition(700.0,1200.0,200.0,700.0);

  // open a window for the program
  winopen("AquaRobot");

  // make a title
  wintitle("AquaRobot");

  // put the IRIS into double buffer mode
  doublebuffer();
```

```
  // configure the IRIS (means use the above command settings)
  gconfig();

  // define acceptable queues
  qdevice(REDRAW);
  qdevice(AKEY);
  qdevice(PKEY);
  qdevice(RKEY);
  qdevice(MENUBUTTON);

  // initial location of viewpoint (camera eye)
  vx = 0.0;
  vy = 40.0;
  vz = 400.0;

  // initial location of robot foot pads
  rfx = 0.0;
  rfy = 0.0;
  rfz = -200.0;

}


// ************************************************************
// Function Make_the_Menus
// ************************************************************

long makethemenus()
  {

     long topmenu;
     long cameramenu;

     // camera views
     cameramenu = newpup();
     addtopup(cameramenu,"Camera View %t ");
     addtopup(cameramenu,"ABOVE %x2 | BEHIND %x3");
     addtopup(cameramenu,"RIGHT SIDE  %x4 | LEFT SIDE %x5");


     // build the top level menu
     topmenu= defpup("Roll Off Side %t| Camera %x1 %m|
                      FileRead %x6 |ResetFile %x8|KeybdRead %x7|
                      Reset %x14| Exit %x15",cameramenu);

     // return the name of this menu
     return(topmenu);

  }


// ************************************************************
// Function Process_Menu_Hit
// ************************************************************

void processmenuhit(long hititem)
  {
    switch (hititem)
       {
          case CAMERA:
                        break;
```

```
        case ABOVE:
                        vx  =   0.0;
                        vy  = 300.0;
                        vz  =   0.0;
                        rfx =   0.0;
                        rfy =   0.0;
                        rfz =   0.0;
                        break;


        case BEHIND:
                        vx  =     0.0;
                        vy  =    50.0;
                        vz  =   250.0;
                        rfx =     0.0;
                        rfy =    30.0;
                        rfz =  -200.0;
                        break;

        case RTSIDE:
                        vx  = 250.0;
                        vy  =  50.0;
                        vz  =   0.0;
                        rfx = -200.0;
                        rfy =  50.0;
                        rfz =   0.0;
                        break;

        case LTSIDE:
                        vx  = -250.0;
                        vy  = 30.0;
                        vz  =    0.0;
                        rfx =  0.0;
                        rfy =   0.0;
                        rfz = -200.0;
                        break;

        case FILEREAD:
                        qenter(PKEY,1);
                        break;

        case RESETFILE:
                        qenter(RKEY,1);
                        break;

        case RESET:
                        vx  = 0.0;
                        vy  = 40.0;
                        vz  = 400.0;
                        rfx = 0.0;
                        rfy = 0.0;
                        rfz = -211.0;
                        break;

        case EXIT:
                        exit(0);
                        break;
    } // End Switch

}
```

```
// ********************************************************
// FUNCTION: BUILDNONMOVINGVIEWINGMATRIX
// PURPOSE: use with objects that are in the same coordinate
//          system and aren't moving with continuous
//          rotations/translations/scalings
// ********************************************************

void buildnonmovingviewingmatrix(float vx,float vy,float vz,
             float refx,float refy,float refz)
{
   loadunit();

   projectionandviewingmatrix(vx,vy,vz,refx,refy,refz);

}


// ********************************************************
// FUNCTION: PROJECTIONANDVIEWINGMATRIX
// PURPOSE:  provides the projection and viewing matrix
// ********************************************************
void projectionandviewingmatrix(float vx,float vy,float vz,float refx,float refy,float ref
z)

[

  // perspective projection 3D for the world coord sys
  // the near and far values are distances from the viewer
  // to the near and far clipping planes.
  // We are at (vx,vy,vz) and looking towards
  // the center point of the object..
  //  (towards (refx,refy,refz)).

  perspective(450,1.25,NEARCLIPPING,FARCLIPPING);
  lookat(vx,vy,vz,refx,refy,refz,0);

}
// ********************************************************
// FUNCTION: LOADUNIT
// PURPOSE:  this routine loads a unit matrix onto the top
//           of the stack
// ********************************************************

void loadunit()
{
  static float  un[4][4] = { 1.0, 0.0, 0.0, 0.0,
                             0.0, 1.0, 0.0, 0.0,
                             0.0, 0.0, 1.0, 0.0,
                             0.0, 0.0, 0.0, 1.0 };

  loadmatrix(un);

}

// ********************************************************
// FUNCTION: AQUA DRAWING
// PURPOSE:  draws the robot at coordinates provided
// ********************************************************

void drawaqua(double *bodyc, double *leg1c, double *leg2c, double
        *leg3c, double *leg4c, double *leg5c, double *leg6c)
{
color(WHITE);
linewidth(3);
```

```
// +x to right, +y up, +z out of screen ->for graphics
// +x out of leg1, +y out of screen, -z down->for kinematics


//          x       z         y
  move(bodyc[3],-bodyc[5],bodyc[4]);
  draw(bodyc[6],-bodyc[8],bodyc[7]);
  draw(bodyc[9],-bodyc[11],bodyc[10]);
  draw(bodyc[12],-bodyc[14],bodyc[13]);
  draw(bodyc[15],-bodyc[17],bodyc[16]);
  draw(bodyc[18],-bodyc[20],bodyc[19]);
  draw(bodyc[3],-bodyc[5],bodyc[4]);

// draws a line from body center to leg1 joint 1
  linewidth(1);
  move(bodyc[0],-bodyc[2],bodyc[1]);
  draw(bodyc[3],-bodyc[5],bodyc[4]);

// draws leg1
  color(YELLOW);
  linewidth(5);
//        x         z         y
  move(leg1c[0],-leg1c[2],leg1c[1]);
  draw(leg1c[3],-leg1c[5],leg1c[4]);
  draw(leg1c[6],-leg1c[8],leg1c[7]);
  draw(leg1c[9],-leg1c[11],leg1c[10]);

// draws leg2
   color(GREEN);
   linewidth(5);
  move(leg2c[0],-leg2c[2],leg2c[1]);
  draw(leg2c[3],-leg2c[5],leg2c[4]);
  draw(leg2c[6],-leg2c[8],leg2c[7]);
  draw(leg2c[9],-leg2c[11],leg2c[10]);

// draws leg3
   color(GREEN);
   linewidth(5);
   move(leg3c[0],-leg3c[2],leg3c[1]);
   draw(leg3c[3],-leg3c[5],leg3c[4]);
   draw(leg3c[6],-leg3c[8],leg3c[7]);
   draw(leg3c[9],-leg3c[11],leg3c[10]);

// draws leg4
   color(GREEN);
   linewidth(5);
   move(leg4c[0],-leg4c[2],leg4c[1]);
   draw(leg4c[3],-leg4c[5],leg4c[4]);
   draw(leg4c[6],-leg4c[8],leg4c[7]);
   draw(leg4c[9],-leg4c[11],leg4c[10]);

// draws leg5
   color(GREEN);
   linewidth(5);
   move(leg5c[0],-leg5c[2],leg5c[1]);
   draw(leg5c[3],-leg5c[5],leg5c[4]);
   draw(leg5c[6],-leg5c[8],leg5c[7]);
   draw(leg5c[9],-leg5c[11],leg5c[10]);

// draws leg6
   color(GREEN);
   linewidth(5);
   move(leg6c[0],-leg6c[2],leg6c[1]);
```

```
    draw(leg6c[3],-leg6c[5],leg6c[4]);
    draw(leg6c[6],-leg6c[8],leg6c[7]);
    draw(leg6c[9],-leg6c[11],leg6c[10]);

}
```

```
// ************************************************************
// FILENAME: AquaLeg.H
// PURPOSE: Declarations for AquaLeg claaa
//
// AUTHOR: S L Davidson
// DATE:  17 Feb 93
// COMMENTS: Definition of AquaLeg claas and functions that
//       apply to this claas
// ************************************************************

#ifndef H_AQUALEG
#define H_AQUALEG

#include <stdio.h>
#include "AquarobotBody.H"
#include "Link.H"
#include "Link0.H"
#include "Link1.H"
#include "Link2.H"
#include "Link3.H"

class AquaLeg
{
 public:

 // these dependent objects are instantiated
 Link0 *link0;
 Link1 *link1;
 Link2 *link2;
 Link3 *link3;

 // the flag is set to 1 if the motion is completed without
 // reaching any link limits
 int motion_complete_flag;

 // the flag is set to 1 if the leg is on the ground
 int leg_support_flag;

 // the angle off of leg one where the leg is attached to
 // the body
 double leg_attachment_angle;


 AquaLeg(AquarobotBody&, double); // constructor and initializer
 ~AquaLeg();                      // destructor
 void MoveIncremental(AquarobotBody &, double delta1, double delta2,
      double delta3);

 double GetLegAttachmentAngle() { return leg_attachment_angle;}
 int GetMotionCompleteFlag() { return motion_complete_flag;}
 void SetLegAttachmentAngle(double angle) {leg_attachment_angle = angle;}
 void SetMotionCompleteFlag(int flag) {motion_complete_flag = flag;}
 int GetLegSupportFlag() { return leg_support_flag;}
 void SetLegSupportFlag(int flag) {leg_support_flag = flag;}

};
#endif
```

```
// ****************************************************************
// FILENAME: AquaLeg.C
// PURPOSE: Implementation of AquaLeg class
// CONTAINS: AquaLeg()
//           Initialize(AquaLeg&, AquarobotBody&)
//           TakePicture(Camera&, AquaLeg&)
//           MoveIncremental (Aqualeg&, deltal,delta2,delta3)
// AUTHOR:  S L Davidson
// DATE:    17 Feb 93
// ****************************************************************
#include "AquaLeg.H"

// ********************************************************
// FUNCTION: ~AquaLeg()
// PURPOSE: destructor of AquaLeg class
// ********************************************************

AquaLeg::~AquaLeg()
{
  delete link0;
  delete link1;
  delete link2;
  delete link3;
}

// ********************************************************
// FILENAME: AquaLeg
// PURPOSE:  constructor of AquaLeg claas
// RETURNS:  AquaLeg class with values
// ********************************************************

AquaLeg::AquaLeg(AquarobotBody &body, double angle)
{
  motion_complete_flag = 1;     // initializes flag value
  SetLegAttachmentAngle(angle);
  link0 = new Link0;
  link1 = new Link1;
  link2 = new Link2;
  link3 = new Link3;


// initial link values initialized

// temp matrix adds in the T_matrix needed for the physical
// attachment of the leg to the body
   matrix temp;

// updates the Transformation matrix from body center to the
// leg attachment point
   temp.UpdateTMatrix(GetLegAttachmentAngle(),0.,0.,0.);
   temp = *body.H_matrix * temp;
   link0->RotateLink(&temp ,link0->GetInboardJointAngle());

   link1->RotateLink(link0->H_matrix, link1->GetInboardJointAngle());
   link2->RotateLink(link1->H_matrix,link2->GetInboardJointAngle());
   link3->RotateLink(link2->H_matrix,link3->GetInboardJointAngle());
}
// ********************************************************
// FILENAME: MoveIncremental
// PURPOSE: calculate the new link values as a leg rotates
// RETURNS:  rotated link's new values are placed in the
//           respective leg's slots
// ********************************************************
```

```
void AquaLeg::MoveIncremental(AquarobotBody &body,double delta1,
              double delta2,double delta3)
 {
 double b;

// set all limit flags to zero
 link1->SetMotionLimitFlag(0);
 link2->SetMotionLimitFlag(0);
 link3->SetMotionLimitFlag(0);

// temp matrix adds in the T_matrix needed for the physical
// attachment of the leg to the body
 matrix temp;
 temp.UpdateTMatrix(GetLegAttachmentAngle(),0.,0.,0.);
 temp = *body.H_matrix * temp;
 link0->RotateLink(&temp,link0->GetInboardJointAngle());

 b = delta1 + link1->GetInboardJointAngle();
 link1->SetInboardJointAngle(b);
 link1->RotateLink(link0->H_matrix,link1->GetInboardJointAngle());

 b = delta2 + link2->GetInboardJointAngle();
 link2->SetInboardJointAngle(b);
 link2->Rotate(link1->H_matrix,link2->GetInboardJointAngle());

 b = delta3 + link3->GetInboardJointAngle();
 link3->SetInboardJointAngle(b);
 link3->RotateLink(link2->H_matrix,link3->GetInboardJointAngle());

 // the motion_complete_flag is set to 1 if the
 // motion_limit_flags on all legs are not set
 SetMotionCompleteFlag(!(link1->GetMotionLimitFlag() ||
   link2->GetMotionLimitFlag() || link3->GetMotionLimitFlag()));

 // prints the status of the requested motion and prints which
 // link's motion_limit_flag was set (if any).
 if (GetMotionCompleteFlag() == 0)
    {printf("Motion Not Completed\n");
     if (link1->GetMotionLimitFlag() == 1)
        printf("link 1 limit exceeded\n");
     if (link2->GetMotionLimitFlag() == 1)
        printf("link 2 limit exceeded\n");
     if (link3->GetMotionLimitFlag() == 1)
        printf("link 3 limit exceeded\n");
    }
  else printf("Motion completed\n");

}
```

```
// ************************************************************
// FILENAME: AquarobotBody.H
// PURPOSE: Declaration of AquarobotBody class
//          Subclass of RigidBody class
// AUTHOR: S L Davidson
// DATE: 20 Sep 92
// ************************************************************

#ifndef H_AQUAROBOTBODY
#define H_AQUAROBOTBODY

#include <stdio.h>
#include "RigidBody.H"
#include "MatrixMy.H"

class AquarobotBody : public RigidBody
{
 public:
 matrix *body_list; // defines the size of the body using coordinates

 double azimuth, elevation, roll;

 AquarobotBody();   // constructor
 void MoveIncremental(double,double, double, double, double, double);

};

#endif
```

```
// ****************************************************************
// FILENAME: AquarobotBody.C
// PURPOSE: Implementation of the AquarobotBody class
// CONTAINS: initializes the body form
// AUTHOR:  S L Davidson
// DATE:    17 Nov 92
// ****************************************************************
#include "AquarobotBody.H"

// ****************************************************************
// FUNCTION: AquarobotBody()
// PURPOSE: constructor of the AquarobotBody class
// RETURNS: AquarobotBody class with values
// ****************************************************************

AquarobotBody::AquarobotBody()
 {
  // each row is a body point (x,y,z)
  // the first (0 row) is the body's physical center, and the rest
  // are the six points of the body
  body_list = new matrix(7,4,0.0);

 // the body's coordinates are defined centered at 0,0,0
 body_list->val(0,0) = 0.; body_list->val(0,1) = 0.;
 body_list->val(0,2) = 0.; body_list->val(0,3) = 1.;
 body_list->val(1,0) = 37.5; body_list->val(1,1) = 0.;
 body_list->val(1,2) = 0.; body_list->val(1,3) = 1.;
 body_list->val(2,0) = 18.75; body_list->val(2,1) = 32.48;
 body_list->val(2,2) = 0.; body_list->val(2,3) = 1.;
 body_list->val(3,0) = -18.75; body_list->val(3,1) = 32.48;
 body_list->val(3,2) = 0.; body_list->val(3,3) = 1.;
 body_list->val(4,0) = -37.5; body_list->val(4,1) = 0.;
 body_list->val(4,2) = 0.; body_list->val(4,3) = 1.;
 body_list->val(5,0) = -18.75; body_list->val(5,1) = -32.48;
 body_list->val(5,2) = 0.; body_list->val(5,3) = 1.;
 body_list->val(6,0) = 18.75; body_list->val(6,1) = -32.48;
 body_list->val(6,2) = 0.; body_list->val(6,3) = 1.;

 // defines the initial location of the body using the
 // H-matrix the inputs to the function are:
 //                  (azimuth, elevation, roll, x, y, z)
 H_matrix->HomogeneousTransform(0.,0.,0.,0., -54.1819);

 // moves the body coordinates to the initial location desired
 body_list->TransformList(*H_matrix,*body_list);

 }

// ****************************************************************
// FUNCTION: MoveIncremental
// PURPOSE: the body is moved based upon commanded incremental
//          degrees of change that are passed in
// ****************************************************************

void AquarobotBody::MoveIncremental(double delaz, double delel,
       double delrol, double delx,double dely, double delz)
 {
 double az, el, ro, x, y, z;
 az = azimuth + delaz;
 el = elevation + delel;
 ro = roll + delrol;
 x = body_list->val(0,0) + delx;
 y = body_list->val(0,1) + dely;
 z = body_list->val(0,2) + delz;
```

```
// only changes are used below since body_list is at current position
 H_matrix->HomogeneousTransform(delaz, delel, delrol, delx, dely, delz);
 body_list->TransformList(*H_matrix,*body_list);

// puts all info in H_matrix
 H_matrix->HomogeneousTransform(az,el,ro,x,y,z);

}
```

```
// ********************************************************
// FILENAME: Kinematica.C
// PURPOSE: to determine positions(x,y,z) from the H_matrix
//        : to read from a file the new link angle changes
//             and update the appropiate leg/link values
//        : to pass items to the gait function
// AUTHOR: S L Davidson
// DATE: 15 February 1993
// ********************************************************

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "MatrixMy.H"
#include "AquaLeg.H"
#include "AquarobotBody.H"
#include "Link.H"

#define GROUNDELEVATION 0.0

// structure designed to receive file input
struct Passing_Items {
   int legnum;
   int body;
   double del1;
   double del2;
   double del3;
   double del4;
   double del5;
   double del6;
 };

// structure recieves next desired robot motion from gait
// planning functions
struct Next_Motion {

//  desired joint increment values returned from the gait function
    double bodymotion[6];
    double leg1motion[3];
    double leg2motion[3];
    double leg3motion[3];
    double leg4motion[3];
    double leg5motion[3];
    double leg6motion[3];

//  actual position status values sent to gait function
    double leg_contact_flag[6];
    double joint_limit_flag[18];
    double foot_1_coord[3];
    double foot_2_coord[3];
    double foot_3_coord[3];
    double foot_4_coord[3];
    double foot_5_coord[3];
    double foot_6_coord[3];
    double body_center_coord[6];

};


// structure designed to consolidate the xyz coordinates of the robot
struct Return_Coordinates {
   double bodyc[21];
   double legic[12];
```

```
   double leg2c[12];
   double leg3c[12];
   double leg4c[12];
   double leg5c[12];
   double leg6c[12];
   int motion_limit_flag[18];
   int leg_support_flag[6];

 };

// xyz coordinates are determined from the H_matrix  and
// return the Cartesian coordinates using the Return_Coordinates structure
Return_Coordinates FindPositions(AquarobotBody &body, AquaLeg &leg1,
      AquaLeg &leg2,AquaLeg &leg3, AquaLeg &leg4, AquaLeg &leg5,
      AquaLeg &leg6)
  {
     Return_Coordinates *rc;
     rc = new Return_Coordinates;
     // body center coordinates
      rc->bodyc[0] =  body.body_list->val(0,0);
      rc->bodyc[1] =  body.body_list->val(0,1);
      rc->bodyc[2] =  body.body_list->val(0,2);
     // body points to draw
      rc->bodyc[3] = body.body_list->val(1,0);
      rc->bodyc[4] = body.body_list->val(1,1);
      rc->bodyc[5] = body.body_list->val(1,2);
      rc->bodyc[6] = body.body_list->val(2,0);
      rc->bodyc[7] = body.body_list->val(2,1);
      rc->bodyc[8] = body.body_list->val(2,2);
      rc->bodyc[9] = body.body_list->val(3,0);
      rc->bodyc[10] = body.body_list->val(3,1);
      rc->bodyc[11] = body.body_list->val(3,2);
      rc->bodyc[12] = body.body_list->val(4,0);
      rc->bodyc[13] = body.body_list->val(4,1);
      rc->bodyc[14] = body.body_list->val(4,2);
      rc->bodyc[15] = body.body_list->val(5,0);
      rc->bodyc[16] = body.body_list->val(5,1);
      rc->bodyc[17] = body.body_list->val(5,2);
      rc->bodyc[18] = body.body_list->val(6,0);
      rc->bodyc[19] = body.body_list->val(6,1);
      rc->bodyc[20] = body.body_list->val(6,2);

   // prints out body coordinates
    printf("body center %3f , %3f , %3f \n",rc->bodyc[0],rc->bodyc[1],
          rc->bodyc[2]);
    printf("body pt 1  %3f, %3f, %3f \n",rc->bodyc[3],rc->bodyc[4],
          rc->bodyc[5]);
    printf("body pt 2 %3f, %3f, %3f \n",rc->bodyc[6],rc->bodyc[7],
          rc->bodyc[8]);
    printf("body pt 3 %3f, %3f, %3f \n",rc->bodyc[9],rc->bodyc[10],
          rc->bodyc[11]);
    printf("body pt 4 %3f, %3f, %3f \n",rc->bodyc[12],rc->bodyc[13],
          rc->bodyc[14]);
    printf("body pt 5 %3f, %3f, %3f \n",rc->bodyc[15],rc->bodyc[16],
          rc->bodyc[17]);
    printf("body pt 6 %3f, %3f, %3f \n\n",rc->bodyc[18],rc->bodyc[19],
          rc->bodyc[20]);


    // joint one leg coordinates: [0]=x [1]=y [2]=z
    rc->leg1c[0] = leg1.link0->H_matrix->val(0,3);
    rc->leg2c[0] = leg2.link0->H_matrix->val(0,3);
    rc->leg3c[0] = leg3.link0->H_matrix->val(0,3);
    rc->leg4c[0] = leg4.link0->H_matrix->val(0,3);
```

```
    rc->leg5c[0] = leg5.link0->H_matrix->val(0,3);
    rc->leg6c[0] = leg6.link0->H_matrix->val(0,3);
    rc->leg1c[1] = leg1.link0->H_matrix->val(1,3);
    rc->leg2c[1] = leg2.link0->H_matrix->val(1,3);
    rc->leg3c[1] = leg3.link0->H_matrix->val(1,3);
    rc->leg4c[1] = leg4.link0->H_matrix->val(1,3);
    rc->leg5c[1] = leg5.link0->H_matrix->val(1,3);
    rc->leg6c[1] = leg6.link0->H_matrix->val(1,3);

    rc->leg1c[2] = leg1.link0->H_matrix->val(2,3);
    rc->leg2c[2] = leg2.link1->H_matrix->val(2,3);
    rc->leg3c[2] = leg3.link1->H_matrix->val(2,3);
    rc->leg4c[2] = leg4.link1->H_matrix->val(2,3);
    rc->leg5c[2] = leg5.link1->H_matrix->val(2,3);
    rc->leg6c[2] = leg6.link1->H_matrix->val(2,3);

    // joint 2 x,y,z coordinates [3]=x [4]=y [5]=z
    rc->leg1c[3] = leg1.link1->H_matrix->val(0,3);
    rc->leg2c[3] = leg2.link1->H_matrix->val(0,3);
    rc->leg3c[3] = leg3.link1->H_matrix->val(0,3);
    rc->leg4c[3] = leg4.link1->H_matrix->val(0,3);
    rc->leg5c[3] = leg5.link1->H_matrix->val(0,3);
    rc->leg6c[3] = leg6.link1->H_matrix->val(0,3);
    rc->leg1c[4] = leg1.link1->H_matrix->val(1,3);
    rc->leg2c[4] = leg2.link1->H_matrix->val(1,3);
    rc->leg3c[4] = leg3.link1->H_matrix->val(1,3);
    rc->leg4c[4] = leg4.link1->H_matrix->val(1,3);
    rc->leg5c[4] = leg5.link1->H_matrix->val(1,3);
    rc->leg6c[4] = leg6.link1->H_matrix->val(1,3);

    rc->leg1c[5] = leg1.link1->H_matrix->val(2,3);
    rc->leg2c[5] = leg2.link1->H_matrix->val(2,3);
    rc->leg3c[5] = leg3.link1->H_matrix->val(2,3);
    rc->leg4c[5] = leg4.link1->H_matrix->val(2,3);
    rc->leg5c[5] = leg5.link1->H_matrix->val(2,3);
    rc->leg6c[5] = leg6.link1->H_matrix->val(2,3);

    // joint motion_limit_flag
    rc->motion_limit_flag[0] =leg1.link1->GetMotionLimitFlag();
    rc->motion_limit_flag[3] = leg2.link1->GetMotionLimitFlag();
    rc->motion_limit_flag[6] = leg3.link1->GetMotionLimitFlag();
    rc->motion_limit_flag[9] = leg4.link1->GetMotionLimitFlag();
    rc->motion_limit_flag[12] = leg5.link1->GetMotionLimitFlag();
    rc->motion_limit_flag[15] = leg6.link1->GetMotionLimitFlag();

    // joint 3 xyz coordinates [6]=x [7]=y [8]=z
    rc->leg1c[6] = leg1.link2->H_matrix->val(0,3);
    rc->leg2c[6] = leg2.link2->H_matrix->val(0,3);
    rc->leg3c[6] = leg3.link2->H_matrix->val(0,3);
    rc->leg4c[6] = leg4.link2->H_matrix->val(0,3);
    rc->leg5c[6] = leg5.link2->H_matrix->val(0,3);
    rc->leg6c[6] = leg6.link2->H_matrix->val(0,3);

    rc->leg1c[7] = leg1.link2->H_matrix->val(1,3);
    rc->leg2c[7] = leg2.link2->H_matrix->val(1,3);
    rc->leg3c[7] = leg3.link2->H_matrix->val(1,3);
    rc->leg4c[7] = leg4.link2->H_matrix->val(1,3);
    rc->leg5c[7] = leg5.link2->H_matrix->val(1,3);
    rc->leg6c[7] = leg6.link2->H_matrix->val(1,3);

    rc->leg1c[8] = leg1.link2->H_matrix->val(2,3);
    rc->leg2c[8] = leg2.link2->H_matrix->val(2,3);
    rc->leg3c[8] = leg3.link2->H_matrix->val(2,3);
    rc->leg4c[8] = leg4.link2->H_matrix->val(2,3);
```

```
    rc->leg5c[8] = leg5.link2->H_matrix->val(2,3);
    rc->leg6c[8] = leg6.link2->H_matrix->val(2,3);

    // joint 3 motion_limit_flag
    rc->motion_limit_flag[1] = leg1.link2->GetMotionLimitFlag();
    rc->motion_limit_flag[4] = leg2.link2->GetMotionLimitFlag();
    rc->motion_limit_flag[7] = leg3.link2->GetMotionLimitFlag();
    rc->motion_limit_flag[10] = leg4.link2->GetMotionLimitFlag();
    rc->motion_limit_flag[13] = leg5.link2->GetMotionLimitFlag();
    rc->motion_limit_flag[16] = leg6.link2->GetMotionLimitFlag();

    // joint 4 xyz coordinates [9]=x [10]=y [11]=z
    rc->leg1c[9] = leg1.link3->H_matrix->val(0,3);
    rc->leg2c[9] = leg2.link3->H_matrix->val(0,3);
    rc->leg3c[9] = leg3.link3->H_matrix->val(0,3);
    rc->leg4c[9] = leg4.link3->H_matrix->val(0,3);
    rc->leg5c[9] = leg5.link3->H_matrix->val(0,3);
    rc->leg6c[9] = leg6.link3->H_matrix->val(0,3);

    rc->leg1c[10] = leg1.link3->H_matrix->val(1,3);
    rc->leg2c[10] = leg2.link3->H_matrix->val(1,3);
    rc->leg3c[10] = leg3.link3->H_matrix->val(1,3);
    rc->leg4c[10] = leg4.link3->H_matrix->val(1,3);
    rc->leg5c[10] = leg5.link3->H_matrix->val(1,3);
    rc->leg6c[10] = leg6.link3->H_matrix->val(1,3);

    rc->leg1c[11] = leg1.link3->H_matrix->val(2,3);
    rc->leg2c[11] = leg2.link3->H_matrix->val(2,3);
    rc->leg3c[11] = leg3.link3->H_matrix->val(2,3);
    rc->leg4c[11] = leg4.link3->H_matrix->val(2,3);
    rc->leg5c[11] = leg5.link3->H_matrix->val(2,3);
    rc->leg6c[11] = leg6.link3->H_matrix->val(2,3);

  // joint 3 motion_limit_flag
    rc->motion_limit_flag[2] = leg1.link3->GetMotionLimitFlag();
    rc->motion_limit_flag[5] = leg2.link3->GetMotionLimitFlag();
    rc->motion_limit_flag[8] = leg3.link3->GetMotionLimitFlag();
    rc->motion_limit_flag[11] = leg4.link3->GetMotionLimitFlag();
    rc->motion_limit_flag[14] = leg5.link3->GetMotionLimitFlag();
    rc->motion_limit_flag[17] = leg6.link3->GetMotionLimitFlag();

// test for supporting legs and adjusting leg_support_flag
    if (fabs(rc->leg1c[11]) >= GROUNDELEVATION) leg1.SetLegSupportFlag(1);
    else leg1.SetLegSupportFlag(0);
    if (fabs(rc->leg2c[11]) >= GROUNDELEVATION) leg2.SetLegSupportFlag(1);
    else leg2.SetLegSupportFlag(0);
    if (fabs(rc->leg3c[11]) >= GROUNDELEVATION) leg3.SetLegSupportFlag(1);
    else leg3.SetLegSupportFlag(0);
    if (fabs(rc->leg4c[11]) >= GROUNDELEVATION) leg4.SetLegSupportFlag(1);
    else leg4.SetLegSupportFlag(0);
    if (fabs(rc->leg5c[11]) >= GROUNDELEVATION) leg5.SetLegSupportFlag(1);
    else leg5.SetLegSupportFlag(0);
    if (fabs(rc->leg6c[11]) >= GROUNDELEVATION) leg6.SetLegSupportFlag(1);
    else leg6.SetLegSupportFlag(0);

// places leg_support_flag into rc
   rc->leg_support_flag[0] = leg1.GetLegSupportFlag();
   rc->leg_support_flag[1] = leg2.GetLegSupportFlag();
   rc->leg_support_flag[2] = leg3.GetLegSupportFlag();
   rc->leg_support_flag[3] = leg4.GetLegSupportFlag();
   rc->leg_support_flag[4] = leg5.GetLegSupportFlag();
   rc->leg_support_flag[5] = leg6.GetLegSupportFlag();

// prints body and leg xyz coordinates
```

```
    int row, col;
    printf("leg1                                          leg2\n");
    for ( row = 1; row<5; row++)
     {
      for (col = 3; col>0; col--)
         printf("%6.4f      ",rc->leg1c[3 * row - col]);
      printf("    ");
      for (col = 3; col>0; col--)
         printf("%6.4f      ",rc->leg2c[3 * row - col]);
      printf("\n");
     }
    printf("\n");
    printf("leg3                                          leg4\n");
    for (row = 1; row<5; row++)
     {
      for (col = 3;col>0; col--)
        printf("%6.4f      ",rc->leg3c[3 * row - col]);
      printf("    ");
      for ( col = 3; col>0; col--)
        printf("%6.4f      ",rc->leg4c[3 * row - col]);
      printf("\n");
     }
     printf("\n");
     printf("leg5                                           leg6\n");
     for (row = 1; row<5; row++)
      {
       for (col = 3; col>0; col--)
        printf("%6.4f      ",rc->leg5c[3 * row - col]);
       printf("    ");
       for (col = 3; col>0; col--)
        printf("%6.4f      ",rc->leg6c[3 * row - col]);
       printf("\n");
      }
      printf("\n");

   return *rc;
}

// ****************************************************
// FUNCTION: File_Use
// PURPOSE: reads desired leg changes from a file
// INPUT: reads from file:
//          format: leg#, delta1, delta2, delta3
// OUTPUT: calculates new leg/link coordinates
// ****************************************************

Passing_Items File_Use(FILE *ifp,AquarobotBody &body,AquaLeg &leg1,
    AquaLeg &leg2,AquaLeg &leg3, AquaLeg &leg4, AquaLeg &leg5,
    AquaLeg &leg6)
{
 Passing_Items *pass;
 pass = new Passing_Items;
 fscanf(ifp,"%d %lf %lf %lf %lf %lf %lf",&pass->body,&pass->del1,
     &pass->del2,&pass->del3,&pass->del4,&pass->del5,&pass->del6);
 if (pass->legnum < 9)
   {

body.MoveIncremental(pass->del1,pass->del2,pass->del3,pass->del4,
     pass->del5,pass->del6);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
  leg1.MoveIncremental(body,pass->del1,pass->del2,pass->del3);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
```

```
  leg2.MoveIncremental(body,pass->del1,pass->del2,pass->del3);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
  leg3.MoveIncremental(body,pass->del1,pass->del2,pass->del3);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
  leg4.MoveIncremental(body,pass->del1,pass->del2,pass->del3);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
  leg5.MoveIncremental(body,pass->del1,pass->del2,pass->del3);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
  leg6.MoveIncremental(body,pass->del1,pass->del2,pass->del3);
 fscanf(ifp,"%d %lf %lf %lf",&pass->legnum,&pass->del1,&pass->del2,
     &pass->del3);
 if (pass->legnum == 0)
      { pass->del1 = 0.0;
        pass->del2 = 0.0;
        pass->del3 = 0.0;
      } ;
  );
 return *pass;
}

// ************************************************************
// FILENAME:  TransferToGait
// PURPOSE: places the body center and leg  Cartesian coordinates
//          in a Next_Motion structure for gait algorithm use
// ************************************************************

Next_Motion TransferToGait(Return_Coordinates &coord, AquarobotBody &body)
{
  Next_Motion *temp;
  temp = new Next_Motion;

  temp->body_center_coord[3] = coord.bodyc[0];  //x
  temp->body_center_coord[4] = coord.bodyc[1];  //y
  temp->body_center_coord[5] = coord.bodyc[2];  //z

  temp->foot_1_coord[0] = coord.leg1c[9];       //x
  temp->foot_1_coord[1] = coord.leg1c[10];      //y
  temp->foot_1_coord[2] = coord.leg1c[11];      //z

  temp->foot_2_coord[0] = coord.leg2c[9];       //x
  temp->foot_2_coord[1] = coord.leg2c[10];      //y
  temp->foot_2_coord[2] = coord.leg2c[11];      //z

  temp->foot_3_coord[0] = coord.leg3c[9];       //x
  temp->foot_3_coord[1] = coord.leg3c[10];      //y
  temp->foot_3_coord[2] = coord.leg3c[11];      //z

  temp->foot_4_coord[0] = coord.leg4c[9];       //x
  temp->foot_4_coord[1] = coord.leg4c[10];      //y
  temp->foot_4_coord[2] = coord.leg4c[11];      //z

  temp->foot_5_coord[0] = coord.leg5c[9];       //x
  temp->foot_5_coord[1] = coord.leg5c[10];      //y
  temp->foot_5_coord[2] = coord.leg5c[11];      //z

  temp->foot_6_coord[0] = coord.leg6c[9];       //x
  temp->foot_6_coord[1] = coord.leg6c[10];      //y
  temp->foot_6_coord[2] = coord.leg6c[11];      //z

// current body elevation
```

```
 temp->body_center_coord[1] = -1. * body.H_matrix->val(2,0);

// current body azimuth
temp->body_center_coord[0] =asin(body.H_matrix->val(1,0) /
        cos(temp->body_center_coord[1]));

// current body roll
temp->body_center_coord[2]=asin(body.H_matrix->val(2,1) /
        cos(temp->body_center_coord[1]));

// joint_limit_flag
for (int i=0; i<17; i++)
    temp->joint_limit_flag[i] = coord.motion_limit_flag[i];

// leg_contact_flag
for (int j=0; j<7; j++)
    temp->leg_contact_flag[j] = coord.leg_support_flag[i];

   return *temp;
}

// ****************************************************
// FUNCTION:gait algorithm
// PURPOSE: to provide a temporary gait function for
//          testing purposes
// ****************************************************

Next_Motion GaitAlgorithm (Next_Motion &in)
{
 Next_Motion *temp;
 temp = new Next_Motion;

 for (int i = 0; i<6; i++)
    temp->bodymotion[i] = 0.0;

 for (i = 0; i<3; i++)
     { temp->leg1motion[i] = 0.0;
       temp->leg2motion[i] = 0.0;
       temp->leg3motion[i] = 0.0;
       temp->leg4motion[i] = 0.0;
       temp->leg5motion[i] = 0.0;
       temp->leg6motion[i] = 0.0;
     };

 //  movement desired

return *temp;
}
```

```
// ************************************************************
// FILENAME: Link.H
// PURPOSE: Declarations for class Link
//
// AUTHOR: S L Davidson
// DATE: 18 Sept 92
// COMMENTS: Definition of Link class
// ************************************************************

#ifndef H_LINK
#define H_LINK

#include <stdio.h>
#include <math.h>
#include "RigidBody.H"
#include "MatrixMy.H"

class Link: public RigidBody
 {

private:

int motion_limit_flag;
double link_length;
double twist_angle;
double inboard_joint_angle;
double inboard_joint_displacement;
double inboard_link;
double min_joint_angle;                 // rotary link
double max_joint_angle;                 // rotary link

public:

Link ( int mlf, double ll, double ta, double ija, double ijd, double il,
       double min_ja, double max_ja );
~Link();

void Rotate(matrix*, double);
void RotateLink(matrix*, double);

int GetMotionLimitFlag() {return motion_limit_flag;}
double GetLinkLength() {return link_length;}
double GetTwistAngle() {return twist_angle;}
double GetInboardJointAngle() {return inboard_joint_angle;}
double GetInboardJointDisplacement() {return inboard_joint_displacement;}
double GetInboardLink() {return inboard_link;}
double GetMinJointAngle() {return min_joint_angle;}
double GetMaxJointAngle() {return max_joint_angle;}

void SetMotionLimitFlag(int a) {motion_limit_flag = a;}
void SetLinkLength(double a) {link_length = a;}
void SetTwistAngle(double a) {twist_angle = a;}
void SetInboardJointAngle(double a) {inboard_joint_angle = a;}
void SetInboardJointDisplacement(double a) {inboard_joint_displacement =a;}
void SetInboardLink(double a) {inboard_link = a;}
void SetMinJointAngle(double a) {min_joint_angle = a;}
void SetMaxJointAngle(double a) {max_joint_angle = a;}

);
#endif
```

```
// ****************************************************
// FILENAME: Link.C
// PURPOSE: Implementation of class Link
// CONTAINS: UpdateAMatrix ()
//           Rotate (double angle)
//           RotateLink (double angle)
// AUTHOR: S L Davidson
// DATE: 18Sept 92
// ****************************************************


#include "Link.H"

const int True = 1;
const int False = 0;



// ****************************************************
// FUNCTION: Link
// PURPOSE:  Constructor for Link
// RETURNS: a link with values
// ****************************************************

Link::Link ( int mlf, double ll, double ta, double ija, double ijd, double il,
       double min_ja, double max_ja )
{
  motion_limit_flag = mlf;
  link_length = ll;
  twist_angle = ta;
  inboard_joint_angle = ija;
  inboard_joint_displacement = ijd;
  inboard_link = il;
  min_joint_angle = min_ja;
  max_joint_angle = max_ja;

  H_matrix->UpdateTMatrix(ija,ta,ll,ijd);
}

// ****************************************************
// FUNCTION: ~Link
// PURPOSE: destructor for Link class
// ****************************************************

Link::~Link()
{
   delete node_list;
}

// ****************************************************
// FUNCTION: Rotate
// PURPOSE: rotates a Link by changing the T Matrix
//       by the inboard joint angle desired
// RETURNS: an updated T_matrix within the Link object
// ****************************************************

void Link::Rotate (matrix *mat, double angle)
{
SetInboardJointAngle(angle);

T_matrix->UpdateTMatrix(GetInboardJointAngle(),GetTwistAngle(),
         GetLinkLength(),GetInboardJointDisplacement());
```

```
//  the "mat" is the inboard link's T matrix (or the body's
//   T matrix for the inboard joint
*H_matrix = *mat * *T_matrix;

}

// ********************************************************
// FUNCTION: RotateLink
// PURPOSE: determines if the rotation is within physical
//     joint constraints. If outside the workspace the min
//     or max limit applicable is used.
//        : this function calls the Rotate function
// RETURNS: sets range of inboard joint angle if desired is
//          outside physical constraints
// ********************************************************

void Link::RotateLink(matrix *mat, double angle)
  {
  double tester;                // temporary variable
  tester = GetMinJointAngle();
  if (angle < tester)
    { angle = tester;
      SetMotionLimitFlag(1);
    }

  tester = GetMaxJointAngle();
  if (angle > tester)
    { angle = tester;
      SetMotionLimitFlag(1);
    }

  Rotate(mat, angle);
}
```

```
// ************************************************************
// FILENAME: Link0.H
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#ifndef H_LINK0
#define H_LINK0

#include "Link.H"

class Link0 : public Link
{

private:

public:

  Link0();

};
#endif
```

```
// ************************************************************
// FILENAME: Link0.C
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#include "Link0.H"

Link0::Link0() : Link ( 0, 37.5, 0.0, 0.0, 0.0, -1.0,
                        -360.0,360.0)
{
  node_list->val(0,3) =1.; node_list->val(1,3) =1.;
  node_list->val(2,0) =37.5; node_list->val(2,3) = 1.;


}
```

```
// ************************************************************
// FILENAME: Link1.H
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#ifndef H_LINK1
#define H_LINK1

#include "Link.H"

class Link1 : public Link
{

private:

public:
        Link1();
};

#endif
```

```
// ************************************************************
// FILENAME: Link1.C
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#include "Link1.H"

Link1::Link1() : Link ( 0, 20.0, -90.0, 66.4, 0.0, 0,-106.6,73.4)
{
  node_list = new matrix(4,4,0.0);
  node_list->val(0,3) =1.;
  node_list->val(1,3) =1.;
  node_list->val(2,0) = 20.0;
  node_list->val(2,3) = 1.;

  T_matrix = new matrix(4,4,0.0);

}
```

```
// ************************************************************
// FILENAME: Link2.H
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#ifndef H_LINK2
#define H_LINK2

#include "Link.H"

class Link2 : public Link
{

private:

public:
        Link2();
};
#endif
```

```
// ************************************************************
// FILENAME: Link2.C
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#include "Link2.H"

Link2::Link2() : Link ( 0, 50.0, 0.0, -156.4, 0.0, 1.0, -156.4, 23.6)
{
  node_list = new matrix(4,4,0.0);
  node_list->val(0,3) =1.; node_list->val(1,3) =1.;
  node_list->val(2,0) =50.; node_list->val(2,3) = 1.;

  T_matrix = new matrix(4,4,0.0);

}
```

```
// ********************************************************
// FILENAME: Link3.H
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ********************************************************

#ifndef H_LINK3
#define H_LINK3

#include "Link.H"

class Link3 : public Link
{

private:

public:
        Link3();
};

#endif
```

```
// ************************************************************
// FILENAME: Link3.C
// PURPOSE: Declarations for class Link0
//
// AUTHOR: S L Davidson
// DATE: 17 Sept 92
// COMMENTS:
// ************************************************************

#include "Link3.H"

Link3::Link3() : Link ( 0, 100.0, 0.0,0.0, 0.0, 2.0, -360.0,360.0)
{
  node_list = new matrix(4,4,0.0);
  node_list->val(0,3) =1.; node_list->val(1,3) =1.;
  node_list->val(2,0) =100.; node_list->val(2,3) = 1.;

  T_matrix = new matrix(4,4,0.0);
```

```
// ********************************************************
// FILENAME: MatrixMy.H
// PURPOSE: To provide for a matrix class to accomplish
//        some necessary robotic and kinematic needs.
// AUTHOR: S L Davidson
// DATE: 29 Oct 92
// COMMENTS: DHMatrix, Homogeneous Transform, and
//     TransformList are included
// ********************************************************

#ifndef H_MATRIX
#define H_MATRIX

const double deg_to_rad = .017453292519943295;

class matrix

 {

 struct matrep
  {
   double **m;
   int r, c, n;
  }*p;

 public:
  matrix(const matrix& x);                    // copy initializer
  ~matrix();                                  // class destructor
  matrix();                                   // class constructor
  matrix(int, int, double );                  // class constructor
  matrix operator=(const matrix& rval);
  matrix operator+(const matrix& rval);
  matrix operator*(const matrix& rval);
  matrix operator*(double);
  double & val(int row, int col)const;        // spot value
  void print();                               // prints matrix

  int rows() const {return p->r;};            // returns number of rows
  int cols() const {return p->c;};            // returns number of columns

 // Craig method used
  matrix & HomogeneousTransform(double,double,double,double,double,double);
  matrix & DHMatrix(double, double, double,double, double, double);
  matrix & UpdateTMatrix(double, double, double, double);
  matrix & TransformList(matrix&, matrix&);

 };

#endif
```

```
// ******************************************
// FILENAME: MatrixMy.C
// PURPOSE:  Implementation of MatrixMy class
// CONTAINS: functions which operate upon matrix
//           type variables
// AUTHOR: S L Davidson
// DATE: 20 Feb 93
// ************************************************

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "MatrixMy.H"
#include "AquaLeg.H"

// ************************************************
// FUNCTION: matrix()
// PURPOSE: constructor of a matrix type
//        : creates a 4 by 4 matrix (by default)
// RETURNS: a matrix with 0.0 in all spaces
// ************************************************
matrix::matrix()
{
 p = new matrep;                // pointer to matrix structure
 p->r = 4;                      // r is number of rows
 p->c = 4;                      // c is number of columns
 p->m = new double *[4];        // m is the value array
                                // m consists of a 4 pointer array
 int x;
 for (x =0; x<4; x++)
   p->m[x] = new double[4];     // produces an array of four
                                // items per array pointer
 p->n = 1;

 int j;
 for (int i=0; i<4; i++)
     for (j=0; j<4; j++)
       p->m[i][j] = 0.0;        // each matrix is given the initial
                                // value of 0.0
 }

// ************************************************************
// FUNCTION: matrix(row, col, initval)
// PURPOSE: constructor of a matrix type
//        : creates a 1 by 1 matrix by default in which all the
//          item values are 0.0 or matrix size and values
//          indicated
// RETURNS: a matrix of sizedesired with initial values desired
// ************************************************************

matrix::matrix(int rows = 1, int col = 1, double initval = 0.)
{
 p = new matrep;                // pointer to matrix structure
 p->r = rows;                   // r is number of rows
 p->c = col;                    // c is number of columns
 p->m = new double *[rows];     // produces the desired number
                                // of rows

 int x;
 for (x =0; x<rows; x++)
   p->m[x] = new double[col]; // each row is given an array equal
                                // to the number of columns desired
 p->n = 1;
```

```
 int j;
 for (int i=0; i<rows; i++)
     for (j=0; j<col; j++)
       p->m[i][j] = initval;  // initializes each value to
                              // desired initval

 )

// ****************************************************************
// FUNCTION: matrix(matrix&)
// PURPOSE:  deep copy constructor of the matrix type
// RETURNS:  a complete identical copy of the matrix
// ****************************************************************

matrix::matrix(const matrix& x)
 (
 x.p->n++;
 p = x.p;
 )

// ****************************************************************
// FUNCTION: operator=
// PURPOSE:  operator overload function of the equals sign
//         :  produces another matrix which points to the original
// RETURNS:  copy of matrix is made in the other one
// ****************************************************************

matrix matrix::operator=(const matrix& rval)
 {
 if (--p->n == 0)
  {
  for (int x=0; x<rows(); x++)
    delete p->m[x];
  delete p->m;
  delete p;
  )
  rval.p->n++;
  p = rval.p;
  return *this;
 )

// ****************************************************************
// FUNCTION: ~matrix()
// PURPOSE: destructor of a matrix type
// ****************************************************************

matrix::~matrix()
 {
 if (--p->n == 0)
  (
   for (int x=0; x<rows(); x++) delete p->m[x];
   delete p->m;
   delete p;
  )
 )

// ****************************************************************
// FUNCTION: val(row, col)
// PURPOSE: finds the value in a matrix given row and column
// RETURNS: value in spot in desired row and column
// ****************************************************************

double & matrix::val(int row, int col) const
```

```
 (
  return (p->m[row][col]);
 )

// ****************************************************************
// FUNCTION: operator*
// PURPOSE: operator overload
//        : provides multiplication of two matrices
// RETURNS: the product of two matrices
// ****************************************************************

matrix matrix::operator*(const matrix& arg)
 {
 matrix result(rows(),arg.cols(),0.0); // temporary matrix conatructed

 for (int row=0; row<rows();row++)
   { int col;
     for (col=0; col<arg.cols(); col++)
     (
      double sum=0.0;
      for (int i=0; i<cols(); i++)
        sum += p->m[row][i] * arg.val(i,col);
      result.val(row,col) = sum;
     )
   )
 return result;
 )

// ****************************************************************
// FUNCTION: operator*(double)
// PURPOSE:  operator overload
//        :  provides multiplication of a scalar and a matrix
// RETURNS:  the matrix product
// ****************************************************************

matrix matrix::operator*(double a)
 (
  matrix result(rows(),cols(),0.0); // temporary matrix constructed

  for (int i=0; i<rows(); i++)
   (
     for (int j=0; j<cols(); j++)
      ( double ans;
        ans = result.val(i,j) * a;
       result.val(i,j) = ans;
      )
   )
  return result;
 )

// ****************************************************************
// FUNCTION: operator+
// PURPOSE:  operator overload
//        :  provides addition of two matrices
// RETURNS:  the matrix sum
// ****************************************************************

matrix matrix::operator+(const matrix& arg)
 {
   matrix sum(rows(),cols(),0.0); // temporary matrix constructed

   for (int i=0; i<rows(); i++)
     { int j;
       for (j=0; j<cols(); j++)
```

```
      sum.p->m[i][j] = p->m[i][j] + arg.val(i,j);
    }

  return sum;
 }

// ************************************************************
// FUNCTION: print()
// PURPOSE:  prints the values of the matrix
// RETURNS:  a print out to the screen of the matrix contents
// ************************************************************

void matrix::print()
 {

  for (int row=0; row<rows(); row++)
    {
     int col;
     for (col=0; col<cols(); col++)
       printf("%6.6f ", p->m[row][col]);
     printf("\n");
    }

 }

// ************************************************************
// FUNCTION: Homogeneous Transform
// PURPOSE: constructs a transformation matrix
// RETURNS: a matrix
// ************************************************************

matrix & matrix::HomogeneousTransform(double azimuth,double elevation,
          double roll, double x,double y, double z)
{
   double spsi = sin(azimuth);
   double cpsi = cos(azimuth);
   double sth = sin(elevation);
   double cth = cos(elevation);
   double sphi = sin(roll);
   double cphi = cos(roll);
   val(0,0) = (cpsi * cth);
   val(0,1) = ((cpsi * sth * sphi) - (spsi * cphi));
   val(0,2) = ((cpsi * sth * cphi) + (spsi * sphi));
   val(0,3) = x;
   val(1,0) = (spsi * cth);
   val(1,1) = ((cpsi * cphi) + (spsi * sth * sphi));
   val(1,2) = ((spsi * sth * cphi) - (cpsi * sphi));
   val(1,3) = y;
   val(2,0) = (-sth);
   val(2,1) = (cth * sphi);
   val(2,2) = (cth * cphi);
   val(2,3) = z;
   val(3,0) = 0.0;
   val(3,1) = 0.0;
   val(3,2) = 0.0;
   val(3,3) = 1.0;

   return *this;

 }

// ************************************************************
// FUNCTION: DH Matrix
// PURPOSE: constructs a DH matrix
```

```
// RETURNS: a matrix
// ***********************************************************

matrix & matrix::DHMatrix(double cosrotate, double sinrotate,
          double costwist, double sintwist, double length,
          double translate)
 {
   val(0,0) = cosrotate;
   val(0,1) = -1 * sinrotate;
   val(0,2) = 0.0;
   val(0,3) = length;
   val(1,0) = sinrotate * costwist;
   val(1,1) = costwist * cosrotate;
   val(1,2) = -1 * sintwist;
   val(1,3) = translate * -1 * sintwist;
   val(2,0) = sintwist * sinrotate;
   val(2,1) = sintwist * cosrotate;
   val(2,2) = costwist;
   val(2,3) = translate * costwist;
   val(3,3) = 1.0;

   return *this;
  }

// ***************************************************************
// FUNCTION: Update T Matrix
// PURPOSE:  constructs a transformation matrix
//         :  calls the DH matrix function
// RETURNS:  a matrix
// ***************************************************************

matrix & matrix::UpdateTMatrix(double rotate_angle, double twist_angle,
         double length, double translation)
 {
 rotate_angle = rotate_angle * deg_to_rad;
 twist_angle = twist_angle * deg_to_rad;
 double cosrotate = cos(rotate_angle);
 double sinrotate = sin(rotate_angle);
 double costwist = cos(twist_angle);
 double sintwist = sin(twist_angle);

 DHMatrix(cosrotate, sinrotate, costwist, sintwist,length,translation);

  return *this;
 }

// ***************************************************************
// FUNCTION: Transform List
// PURPOSE:  transfers coordinates to new position based upon H_matrix
// RETURNS:  a transformed node_list as a matrix
// ***************************************************************

matrix & matrix::TransformList(matrix &H_matrix, matrix &b)
 {
  matrix temp(4,1,0.0);            // temporary matrix constructed

  for (int i = 0; i<b.rows(); i++)
   {
//  transposes the node_list so multiplication can be accomplished
    temp.val(0,0) = b.val(i,0);
    temp.val(1,0) = b.val(i,1);
    temp.val(2,0) = b.val(i,2);
    temp.val(3,0) = b.val(i,3);
```

```
    matrix  middle = H_matrix * temp;

//  transposes the node_list back to original form
    val(i,0) = middle.val(0,0);
    val(i,1) = middle.val(1,0);
    val(i,2) = middle.val(2,0);
    val(i,3) = middle.val(3,0);
   };

    return *this;
 }
```

```
// ******************************************************
// FILENAME: RigidBody.H
// PURPOSE:  construct the superclass for robot systems
// AUTHOR: S L Davidson
// DATE: 18 Sept 92
// ******************************************************

#ifndef H_RIGIDBODY
#define H_RIGIDBODY

#include "MatrixMy.H"

const double gravity = 32.2185;

class RigidBody
{


public:
matrix *node_list;
matrix *H_matrix , *T_matrix;

RigidBody();
~RigidBody();

};
#endif
```

```
// ****************************************************
// FILENAME: RigidBody.C
// PURPOSE:  Implementation of class RigidBody
// CONTAINS: superclass of robot system
//         : common slots initiated
// AUTHOR: S L Davidson
// DATE: 18 Feb 93
// ****************************************************

#include "RigidBody.H"

// ****************************************************
// FUNCTION: RigidBody()
// PURPOSE:  constructor of Rigid Body class
// RETURNS:  produced Rigid Body class
// ****************************************************

RigidBody::RigidBody()
{
 node_list = new matrix(4,4,0.0);
 H_matrix = new matrix(4,4,0.0);
 T_matrix = new matrix(4,4,0.0);
};

// ****************************************************
// FUNCTION: ~RigidBody()
// PURPOSE:  destructor of the class
// ****************************************************

RigidBody::~RigidBody()
{

// delete   node_list;
// delete   H_matrix;

}
```

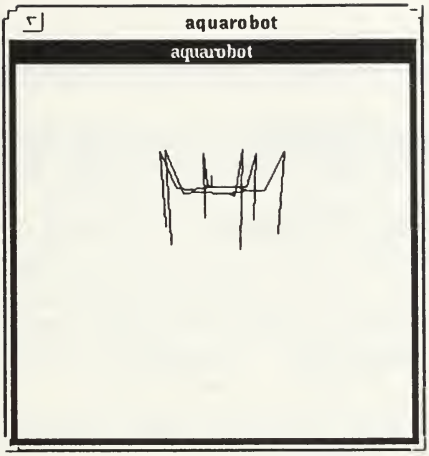
APPENDIX C - CLOS SCRIPT AND GRAPHICS

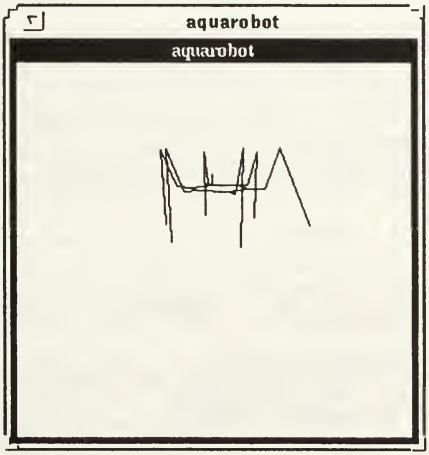
```
typescript                                          1

Script started on Wed Mar 10 08:35:48 1993
hydra% cl
Allegro CL 4.1 [SPARC; R1] (7/8/92 9:07)

;; Copyright Franz Inc., Berkeley, CA, USA
;; Unpublished.  All rights reserved under the copyright laws
;; of the United States.

;; Restricted Rights Legend
;; ------------------------
;; Use, duplication, and disclosure by the Government are subject to
;; restrictions of Restricted Rights for Commercial Software developed
;; at private expense as specified in DOD FAR 52.227-7013 (c) (1) (ii).
;; Optimization settings: safety 1, space 1, speed 1, debug 2
;; For a complete description of all compiler switches given the current
;; optimization settings evaluate (EXPLAIN-COMPILER-SETTINGS).
USER(1): (load "load-files.cl")
; Loading /n/aquarius/work/mcghee/aquarobot/load-files.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/camera.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/link.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/rigid-body.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/robot-kinematics.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/aqua.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/aqua-leg.cl.
;   Loading /n/aquarius/work/mcghee/aquarobot/aqua-link.cl.
T
USER(2): (aqua-picture)
NIL
USER(3): (setf move-list '((0 0 0 0 0 0) (0 0 0) (0 0 0) (.1 .2 .3) (0 0 0)
(0 0 0) (0 0 0)))
((0 0 0 0 0 0) (0 0 0) (0 0 0) (0.1 0.2 0.3) (0 0 0) (0 0 0) (0 0 0))
USER(4): (move-incremental aqua-1 move-list)
T
USER(5): (new-picture)
NIL
USER(6): (exit)
; killing "Default Window Stream Event Handler"
; killing "X11 event dispatcher"
; killing "Initial Lisp Listener"
; Exiting Lisp
hydra% exit
hydra%
script done on Wed Mar 10 08:50:16 1993
```


aquarobot
aquarobot


aquarobot
aquarobot

LIST OF REFERENCES

Akizono, J., et al, "Development on Walking Robot for Underwater Inspection", Advanced Robotics: 1989, Waldron, K. editor, Springer-Verlag, Berlin, Heidelberg, 1989, pp. 652-663.

Ammeraal, L., C++ For Programmers, John Wiley & Sons Ltd., West Sussex, England, 1991.

Anon, Inertial Sensor: General Product Line, Cat. No. 934-2-E 9107-2-F, Tokimec, Inc., Tokyo, Japan, 1992, Gyrocompass TSG-10, p. 13.

Anon, Nature's Technology, videotape, British Broadcasting Corporation, London, England, November 1987.

Bekker, M., Introduction to Terrain-Vehicle Systems, University of Michigan Press, Ann Arbor, Michigan, 1969.

Booch, G., Object Oriented Design With Applications, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1991.

Coplien, J., Advanced C++ Programming Styles and Idioms, AT&T Bell Telephone Laboratories, Incorporated, Menlo Park, California, 1992.

Craig, J., Introduction to Robotics: Mechanics and Control, Second Edition, Addison-Wesley Publishing Company, Inc.,Menlo Park, California, 1989.

de Paula, E. and Nelson, M., "Designing a Class Hierarchy," Proc. of Technology of Object-Oriented Languages & Systems International Conference, Tools USA, Santa Barbra, California, July 29 - August 1, 1991, pp. 203-218.

Eckel, B., Using C++, McGraw-Hill, Inc., Berkeley, California, 1989.

Ferrell, W., and Sheridan, T., "Supervisory Constrol of Remote Manipulation," IEEE Spectrum, v. 4, No. 10, 1967, pp. 81-88.

Fink, A., Object Oriented Programming: An Assessment of Fundamental Concepts and Design Considerations, Masters Thesis, Naval Postgraduate School, Monterey, California, March 1992.

Iwasaki, M., et al., "Development of Aquatic Walking Robot for Underwater Inspection," Report of the Port and Harbour Research Institute, v. 26, No. 5, December 1987, pp. 393-422.

Keene, S., Object-Oriented Programming in Common LISP, Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

Korson, T. and McGregor, J., "Understanding Object-Oriented Programming: A Unifying Paradigm," Communications of the ACM, v. 33, No. 9, September 1990, pp. 40-60.

McGhee, R., et al., "An Approach to Computer Coordination of Motion for Energy-Efficient Walking Machines," Bulletin of Mechanical Engineering Laboratory, No. 43, Mechanical Engineering Laboratory, Ibaraki-ken, Japan, 1986, pp. 1-21.

McGhee, R., "Vehicular Legged Locomotion," Advances in Automation and Robotics, v. 1, JAI Press Inc., Greenwich, Connecticut, 1985, pp. 259-284.

McGhee, R., "Control Needs In Prosthetics and Orthotics," Proceedings of 1977 IEEE Joint Automatic Control Conference, San Francisco, California, June 1977, pp. 567-573.

McMillan, S., Parallel Real-Time Dynamic Simulation of an Underwater Legged Robot, Doctorate Proposal, Ohio State University, Ohio, 24 November 1992.

Pugh, D., An Autopilot for a Terrain-Adaptive Hexapod Vehicle, Masters Thesis, The Ohio State University, Columbus, Ohio, September 1982.

Robison, B., et al., "A Scientific Perspective On the Relative Merits of Manned and Unmanned Vehicles," Proc. of INTERVENTION/ROV '92 Conference & Exposition, San Diego, California, June 10 - 12, 1992, pp. 485-489.

Schue, C., Gait Planning for a Hexapod Underwater Walking Machine, Masters Thesis, Naval Postgraduate School, Monterey, California, June 1993.

Snyder, A., "Encapsulation and Inheritance In Object-Oriented Programming Languages", OOPSLA Conference Proceedings, Portland, Oregon, September 29 - October 2, 1986.

Spong, M. and Vidyasagar, M., Robot Dynamics and Control, John Wiley & Sons, Inc., New York, New York, 1989.

Steele, G., Common LISP, Digital Equipment Corporation, United States of America, 1990.

Stefik, M., and Bobrow, D., "Object-Oriented Programming: Themes and Variations," AI Magazine, Winter 1986, v. 6, No. 4, pp. 40-62.

Stein, J., ed., The Random House College Dictionary, Revised Edition, Random House, Inc., New York, New York, 1979.

Stroustrup, B., The C++ Programming Language, 2nd edition, Addison-Wesley Publishing Company, New York, New York, 1991.

Suzuki, K., and Schue, C., Aquarobot Simulator User's Guide, Department of Computer Science, Naval Postgraduate School, Monterey, California, February 1993.

Takahashi, H., private conversation, Naval Postgraduate School, Monterey, California, January 1993.

Waldron, K., and Song, S., Machines That Walk: The Adaptive Suspension Vehicle, MIT Press, Cambridge, Massachusetts, 1989.

Waldron, K., and McGhee, R., "The Adaptive Suspension Vehicle," IEEE Control Systems Magazine, December 1986, pp. 7-12.

Wegner, P., "Dimensions of Object-Based Language Design," SIGPLAN Notices, v. 22, No. 12, December 1987, pp. 168-182.

Winston, P. and Horn, B., LISP, Third Edition, Addison-Wesley Publishing Company, Inc., 1989.

Wu, T., "Benefits of Abstract Superclasses," Journal of Object-Oriented Programming, v. 3, No. 6, February 1991, pp. 57-61.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, CA 93943

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943

4. Prof. Yutaka Kanayama, Code CS/Ka 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

5. Prof. Sehung Kwak, Code CS/Kw 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

6. Prof. Robert B. McGhee, Code CS/Mz 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

7. Prof. Harold A. Titus, Code EC/Ts 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943

8. Mr. Norman Caplan 1
   Biological and Critical Systems Division
   Engineering Directorate
   National Science Foundation
   1800 G Street, NW
   Washington, DC 20550

9. Mr. Ted G. Davidson 3
   380 Rock Cut Road
   Walden, NY 12586

10. Prof. David E. Orin 1
Department of Electrical Engineering
Ohio State University
2015 Neil Avenue
Columbus, OH 43210

11. Mr. Hidetoshi Takahashi 1
Port and Harbour Research Institute
Ministry of Transport
1-1, 3-Chome, Nagase
Yokosuka, Japan